Tổng hợp & Biên soạn
Ks. ĐINH XUÂN LÂM
VN-GUIDE

GIÁO TRÌNH HỌC VÀ THỰC HÀNH

Microsoft Visual Basic căn bản

LÝ THUYẾT ĐẦY ĐỦ, TRÌNH BÀY NGẮN GỌN, DỄ HIỂU
PHÙ HỢP CHO NGƯỜI HỌC LẬP TRÌNH VISUAL BASIC

BÀI TẬP THỰC HÀNH CÓ HƯỚNG DẪN TỪNG BƯỚC VÀ
BÀI TẬP TỰ LUYỆN ĐA DẠNG, THEO SÁT THỰC TẾ

PHẦN ÔN TẬP BAO GỒM : CÂU HỎI LÍ THUYẾT
CÂU HỎI TRẮC NGHIỆM
BÀI TẬP THỰC HÀNH



NHÀ XUẤT BẢN THỐNG KÊ

Giáo trình

HỌC VÀ THỰC HÀNH

VISUAL BASIC căn bản

Giáo trình

học và thực hành
Visual Basic
căn bản

Tổng hợp và biên soạn: VN–GUIDE
Và *kỹ sư tin học:* ĐINH XUÂN LÂM

NHÀ XUẤT BẢN THỐNG KÊ

Giáo trình học và thực hành
VISUAL BASIC CĂN BẢN

Chịu trách nhiệm xuất bản
CÁT VĂN THÀNH

Biên tập : HẠNH NGUYÊN
Trình bày : THIÊN TRUNG
Vẽ bìa : THIÊN ÂN
Sửa bản in : LỆ HẰNG

NHÀ XUẤT BẢN THỐNG KÊ
98 Thụy Khuê - Tây Hồ - Hà Nội
CN : 16 Nguyễn Huệ, Quận 1, TP. Hồ Chí Minh
ĐT : 8290047

<u>Liên kết xuất bản</u> :

CTY VĂN HÓA MINH TRÍ - NS. VĂN LANG
25 Nguyễn Thị Minh Khai, Q.I, TP.HCM
ĐT : 8.242157 - 8233022 - Fax : 84.8.235079

In 1000 cuốn khổ 14.5x20.5cm tại Xưởng in CN Trung Tâm Hội Chợ Triển Lãm Việt Nam.
Giấy chấp nhận đăng ký KHXB số 92-357/XB-QLXB Cục xuất bản cấp ngày 26.03.2004.
In xong và nộp lưu chiểu quý 2 năm 2004.

Lời giới thiệu

Giáo trình học và thực hành Visual Basic căn bản *hướng dẫn sử dụng Microsoft Visual Basic for Windows Version 6.0. Sau khi đọc xong tập sách này, bạn có thể dùng ngôn ngữ lập trình Visual Basic để thiết kế các trình ứng dụng chạy trong môi trường Windows.*

Visual Basic là một công cụ phát triển phần mềm, nhưng lợi điểm của nó so với các ngôn ngữ lập trình khác là tiết kiệm thời gian và công sức hơn trong lúc xây dựng một ứng dụng.

Khi thiết kế chương trình với Visual Basic, bạn có thể thấy ngay kết quả qua từng thao tác và giao diện chương trình thực hiện. Điều đó cho phép thay đổi dễ dàng và nhanh chóng màu sắc, kích thước, hình dạng, ... của các đối tượng trong ứng dụng.

Tập sách này dành cho những bạn đọc chưa có kinh nghiệm lập trình tiếp cận những kỹ năng lập trình trong ngôn ngữ Visual Basic một cách nhanh nhất. Do đó, một số chi tiết kỹ thuật lập trình mở rộng trong Visual Basic 6.0 sẽ không được đề cập ở đây. Bạn có thể thay đổi, mở rộng những gợi ý trong sách. Tập sách bao gồm 24 bài và phần phụ lục, cuối mỗi bài có một số câu hỏi và bài tập. Bạn nên tự trả lời trước khi coi tới phần giải đáp (Phụ lục D) ở cuối sách.

Khi viết chúng tôi đã hết sức cố gắng để cuốn sách được hoàn chỉnh, song chắc chắn không tránh khỏi thiếu sót, vì vậy mong nhận được sự góp ý của bạn đọc.

Chúc bạn trở thành một lập trình viên giỏi Visual Basic.

VN–GUIDE

Chương I

Bài 1

Giới thiệu về lập trình Visual Basic

- **Tại sao phải viết chương trình?**
- **Khái quát lịch sử lập trình**
- **Lập trình viên bắt đầu từ đâu?**
- **Quá trình cải tiến ngôn ngữ lập trình**
- **Chạy chương trình để xuất kết quả**
- **Khó khăn trong quá trình xử lý lỗi kỹ thuật**
- **Giao diện đồ họa thay đổi mọi thứ**
- **Chuyển tiếp từ BASIC sang Visual Basic**

Nội dung sách đề cập thấu đáo hơn so với một đĩa hướng dẫn thông thường. Tập sách này hướng dẫn sử dụng Microsoft Visual Basic for Windows, vốn là hệ thống thiết kế chương trình trên Windows hoàn hảo. Với Visual Basic, bạn sẽ có ý tưởng tuyệt vời ban đầu (nếu được phép nói như thế) hầu lập trình trong môi trường Visual Basic. Thêm vào đó, còn có những chương trình minh họa, bạn chỉ tốn ít thời gian nhập liệu và tìm hiểu.

Nếu bạn đã từng lập trình trong một ngôn ngữ khác, hãy tạm quên nó đi. Visual Basic không giống ngôn ngữ lập trình đó. Điểm khác biệt chính là Visual Basic mang lại cảm giác thoải mái trong lúc làm việc mà nhiều ngôn ngữ lập trình khác không có. Với Visual Basic, phần lớn các chương trình bạn tạo đều bằng cách nhấp và di chuyển mouse. Thay vì lập trình, thực ra là bạn thiết kế chương trình. Visual Basic là một trong số ít công cụ lập trình có thể giúp bạn thiết kế hiệu quả trong khi tạo chương trình.

Chương này mô tả tất cả những gì liên quan đến lập trình. Bạn hãy xem lướt qua lịch sử lập trình, đặc biệt là sự biến đổi của ngôn ngữ BASIC trong quá trình phát triển máy tính. (Visual Basic có nguồn gốc từ ngôn ngữ BASIC truyền thống.) Hãy sẵn sàng học cách ứng dụng Visual Basic để lập trình.

Ghi chú

Chương này đề cập những kiến thức cơ bản, tầm quan trọng của việc lập trình và những ngôn ngữ lập trình ban đầu. Nội dung chương sách bàn về hai vấn đề. Thứ nhất, thông qua tìm hiểu lịch sử lập trình, bạn sẽ có được khái niệm hoàn hảo hơn về công cụ lập trình hiện nay. Vấn đề thứ hai là đánh giá đầy đủ hơn các chức năng dễ sử dụng và linh hoạt mà Visual Basic cung cấp so với một số ngôn ngữ lập trình khác. Khả năng của Visual Basic vượt xa sức tưởng tượng của những ai dùng nó chỉ cách đây vài năm.

TẠI SAO PHẢI VIẾT CHƯƠNG TRÌNH?

Khái niệm

Nếu muốn máy tính xử lý chính xác, bạn phải viết chương trình. Bản thân máy tính chẳng thể làm gì được cả. Thực ra, máy tính là một cái máy đần độn không hiểu biết. Ngược lại với những gì bạn có thể đọc trong các cuốn truyện khoa học viễn tưởng, máy tính không làm gì hơn ngoài việc thực hiện một cách máy móc các lệnh do lập trình viên cung cấp. Máy tính thật sự không biết suy nghĩ.

Định nghĩa

Chương trình là tập hợp lệnh, chỉ cho máy tính biết chính xác những gì cần thực hiện.

Hiện nay, khi ai đó mua máy tính về đặt trên bàn thì nó sẽ không hề làm gì đến khi anh ta nạp chương trình vào bộ nhớ máy tính và khởi động chương trình. Chỉ có VCR (thiết bị đọc ghi chương trình lên đĩa từ) là không được lập trình để thực hiện công việc như thế. Máy tính cho rằng các lệnh chi tiết chỉ được tìm thấy trong chương trình.

Giả sử bạn cho thuê tài sản và muốn máy tính tìm hồ sơ của người thuê. Máy tính sẽ không trợ giúp bạn chừng nào bạn nạp và chạy chương trình quản lý tài sản cho thuê. Bạn tìm chương trình như thế ở đâu? Có hai cách để có được chương trình máy tính:

- Mua nó và hy vọng rằng chương trình sẽ thực hiện chính xác những gì bạn muốn
- Viết chương trình của riêng bạn

Mua chương trình mà bạn cần thì đó là một việc dễ dàng và nhanh chóng. Hiện nay, hàng ngàn chương trình đang được bày bán. Quả thật, nhiều chương trình vượt ra ngoài những yêu cầu mà bạn có thể hình dung cho việc viết chương trình.

Trường hợp bạn tìm thấy chương trình thực hiện chính xác những gì mong muốn, bạn đang là người dẫn đầu cuộc chơi. Nếu bạn nhận thấy chương trình đáp ứng chính xác các yêu cầu, nên mua chương trình đó. Bởi lẽ, bỏ tiền mua một chương trình luôn rẻ hơn và nhanh hơn bản thân bạn tự viết hoặc thuê lập trình viên viết.

Hãy suy nghĩ đôi chút về điều này: Giả sử có nhiều chương trình được bán hiện nay có khả năng thực hiện hầu như mọi thứ thì tại sao hàng năm các ngôn ngữ lập trình chẳng hạn như Visual Basic vẫn đang tiếp tục sửa đổi những phiên bản trước đây. Câu trả lời thật đơn giản: Người ta mua máy tính để thực hiện công việc mà họ cần. Các công ty không thể đáp ứng xuể công việc kinh doanh với chương trình máy tính. Họ phải tìm hoặc viết chương trình để máy tính xử lý thông tin vào từng công việc đặc thù. Cách duy nhất đảm bảo chương trình phù hợp với yêu cầu của công ty là công ty phải thiết kế chương trình cho riêng mình.

Chương trình được thiết kế cho các doanh nghiệp không chỉ có một. Không có hai nhà quản lý nào lại quản lý tài chính y hệt nhau. Không có hai nhà khoa học nào cần máy tính thực hiện chính xác các phép tính giống nhau. Và cũng không có hai họa sĩ nào sử dụng công cụ vẽ máy tịnh hệt như nhau. Mặc dù người ta mua bảng tính và trình xử lý từ hầu đáp ứng nhu cầu tính toán đa năng, song nhiều người vẫn đòi hỏi chương trình chuyên dụng hơn cho mỗi công việc.

Nét độc đáo trong lập trình máy tính không đơn thuần làm thỏa mãn yêu cầu mà còn đáp ứng nhiều nhu cầu cá nhân. Lập trình máy tính rất lý thú. Chẳng hạn, nhà điêu khắc ngắm nhìn tác phẩm đã hoàn thành, lập trình viên thì tự hào về chương trình mà mình đã viết. Lúc đọc xong cuốn sách này, bạn sẽ nhận được các chương trình viết sẵn nhưng không còn ý nghĩa đến khi bạn viết được chúng. Khi muốn máy tính thực hiện một số việc xác định, bản thân bạn phải có khả năng thiết kế và viết chương trình.

Một số chương trình có thể thay đổi: Phương pháp thứ ba sẽ đưa ra chương trình chính xác hầu vi tính hóa hệ thống kế toán của công ty. Các công ty phần mềm kế toán thường không chỉ bán chương trình kế toán mà còn đính kèm *mã nguồn* (source code) của chương trình đó. Mã nguồn là danh sách lệnh của chương trình. Bằng cách truy xuất mã nguồn, bạn có thể nắm bắt được những gì công ty phần mềm đã viết và sửa đổi hành vi chương trình phù hợp với yêu cầu.

Bằng cách khởi động chương trình chức năng thay vì chương trình hỗn tạp, bạn sẽ tiết kiệm được thời gian lập trình và tiền bạc. Thật không may, hầu như không có công ty phần mềm kế toán nào lại cung cấp mã nguồn. Đa số chương trình bán ra hiện nay đã được *biên dịch* (compile). Sau khi biên dịch mã nguồn, chương trình có thể chạy. Đây là điểm bất lợi không dễ gì thay đổi hành vi của chương trình đã biên dịch. Vì vậy, đối với đa số chương trình, bạn cần chọn lựa giữa việc mua hay tự viết chúng từ các chương trình nhỏ.

Khái niệm mới

Code (mã lệnh) là tên gọi khác của chương trình.

Có nhiều cách viết chương trình máy tính. Trong mục tiếp theo, bạn sẽ học cách xử lý chương trình nhập liệu khi thực hiện thay đổi giao diện người dùng bằng cách nhấp và trỏ mouse. Phần lớn các chương trình đang sử dụng hiện nay được cung cấp dưới dạng danh sách mã lệnh gồm có trang chứa dòng lệnh máy tính nối tiếp nhau. Visual Basic giúp giảm bớt công việc cực nhọc trong quá trình viết mã lệnh – nói rõ hơn là viết chương trình. Visual Basic cho phép bạn di chuyển thành phần và đặt ảnh đồ họa lên màn hình bằng mouse thay vì yêu cầu bạn cung cấp lệnh và mô tả tỉ mỉ như các ngôn ngữ trước Visual Basic.

Củng cố

Không một chương trình nào làm hài lòng tất cả mọi người. Khi công ty bán ra một chương trình, nó phải hài lòng phần lớn người mua. Một số người cần chương trình xử lý theo cách riêng biệt để hoàn thành yêu cầu. Vì thế họ phải viết chương trình cho mình. May mắn thay, Visual Basic sẽ hỗ trợ trong quá trình viết chương trình.

KHÁI QUÁT LỊCH SỬ LẬP TRÌNH

Khái niệm

Máy tính không thể viết chính xác bất kỳ ngôn ngữ nào. Bạn phải học một ngôn ngữ mà máy tính biết trước khi viết chương trình.

Khái niệm mới

Application – ứng dụng còn là một cách gọi khác của chương trình.

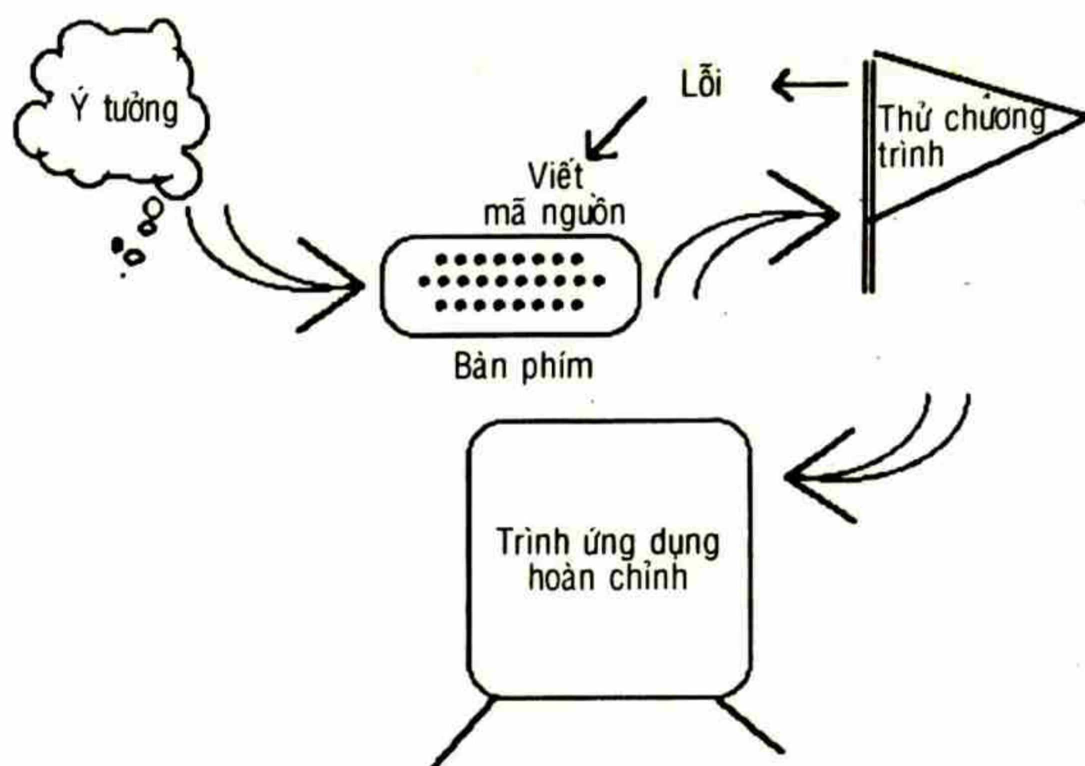
Nhiều người dùng máy tính suốt ngày cho việc xử lý từ, lưu trữ cơ sở dữ liệu và phân tích bảng tính mà không biết những gì đang diễn ra đằng sau hoạt động đó. Bạn phải luôn nhớ rằng máy tính không biết suy nghĩ. Máy tính không biết cách xử lý từ. Nếu muốn máy tính xử lý từ, bạn phải cung cấp lệnh chi tiết dưới dạng chương trình. Chỉ sau khi bạn nạp lệnh chi tiết của chương trình xử lý từ, máy tính mới có thể thực hiện quá trình xử lý từ.

Thật là dễ chịu nếu việc viết chương trình dễ như chỉ thị máy tính thực hiện những gì bạn muốn. Nhiều người có thể sử dụng các lệnh khó hiểu, nhưng máy tính thì không đủ thông minh hiểu hết những yêu cầu mơ hồ đó. Máy tính chỉ có khả năng thi hành lệnh được quy định, và bạn phải cung cấp những lệnh này dưới dạng chương trình. Do đó, buộc phải cung cấp chương trình mà bạn viết. Quá trình viết chương trình, đặc biệt là chương trình phức tạp tốn nhiều thời gian, phải qua nhiều bước khác nhau. Visual Basic đẩy nhanh quá trình tạo chương trình, nhưng thậm chí với Visual Basic, một số chương trình vẫn phải mất thời gian để viết và hoàn tất.

Khái niệm mới

bug ***là lỗi chương trình.***

Hình 1.1 minh họa các bước tiêu biểu mà đa số lập trình viên phải thực hiện khi viết chương trình. Bước thứ nhất, bạn phải có khái niệm về chương trình. Tiếp đến, sử dụng hệ thống thiết kế chương trình như Visual Basic để viết chương trình. Những sai sót hay lỗi kỹ thuật thường xuất hiện trong chương trình ngay cả với chương trình đơn giản nhất. Vì vậy, bạn phải kiểm tra toàn bộ chương trình và hiệu chỉnh lỗi. Quá trình hiệu chỉnh lỗi gọi là *gỡ rối* (debugging). Một khi chương trình không còn lỗi nào, lúc đó bạn sẽ có trình ứng dụng hoàn chỉnh.



Hình 1.1. *Viết chương trình bao gồm nhiều bước.*

Bước thứ hai, viết các mã nguồn, vốn là công việc nhàm chán nhất trong lập trình. Nhớ rằng mã nguồn là lệnh chương trình thực sự mà máy tính sẽ thực hiện. Bạn sẽ tốn nhiều thời gian trong quá trình viết mã nguồn hầu kết thúc chương trình máy tính theo ý thích.

Có nhiều cách viết mã nguồn cho chương trình. Mặc dù máy tính hiện nay có dung lượng bộ nhớ nhiều hơn và vận hành nhanh hơn máy tính trước đây, nhưng máy tính hiện nay cũng không thông minh hơn máy tính thế hệ đầu tiên. Vào những năm cuối thập niên 40, lập trình viên vẫn phải viết chương trình cho máy tính như hiện nay. Điểm khác biệt là lập trình viên hiện nay viết chương trình bằng các công cụ lập trình hoàn hảo hơn như Visual Basic chẳng hạn. Chúng cho phép bạn thiết kế chương trình nhanh hơn trước và tốn ít công sức.

LẬP TRÌNH VIÊN BẮT ĐẦU TỪ ĐÂU?

Máy tính trước đây không được lập trình thông qua bàn phím và mouse. Vì một số máy tính thế hệ đầu tiên thậm chí không có bàn phím! Máy tính này phải lập trình bằng các thành phần của phần cứng. Cho nên trước đây người ta gọi kỹ sư điện tử lập trình chứ không phải gọi là lập trình viên như bây giờ.

Những phương thức lập trình trong phần cứng thường rất khó. Giả như cần thực hiện phép tính khác, buộc bạn phải thay đổi thiết bị

điện tử trong phần cứng máy tính. Lập trình viên phải có kinh nghiệm về ngành công nghệ điện tử mới chỉ thị máy tính vận hành được. Vì vậy, cần phải có phương pháp đẩy nhanh tốc độ lập trình.

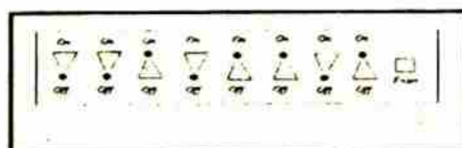
Máy tính trước đây có bộ nhớ tương tự như máy tính hiện nay. Điểm khác biệt là bộ nhớ trước đây rất nhỏ, ngay cả máy tính lớn nhất cũng chỉ có vài trăm địa chỉ nhớ để lưu trữ dữ liệu. Cho dù dung lượng bộ nhớ ít, song các chuyên gia máy tính cũng đã thiết kế được bộ nhớ dùng để lưu trữ dữ liệu cùng với mã lệnh chỉ thị máy tính thi hành tác vụ. Trước khi chưa có bộ nhớ, "mã lệnh" được chứa trong mạch linh kiện phần cứng máy tính.

Mách nước

Bằng cách đặt lệnh điều khiển máy tính vào bộ nhớ cùng với dữ liệu, chương trình máy tính dễ dàng thay đổi hơn, bởi lẽ bộ nhớ có thể thay thế. Các kỹ sư không phải thay đổi thiết bị phần cứng máy tính mỗi khi chuyển đổi chương trình.

Nhà khoa học máy tính đã lắp bảng điều khiển trên máy tính trước đây. Hình 1.2 minh họa bảng điều khiển đơn giản. Lập trình viên lần lượt nhấn công tắc trong hàng để chọn một trong hai trạng thái *bật* (On) hoặc *tắt* (Off), nhấn phím Enter, và lặp lại quy trình này đến khi dãy lệnh "chương trình" sau đây xuất hiện trong bộ nhớ máy tính:

On Off On On Off Off
Off On
Off On On Off On On Off Off
Off Off Off On On Off On Off
On On On On Off Off Off On
Off Off On Off Off On On Off
On On Off On On Off Off On
Off Off On Off On On On Off



Hình 1.2. *Bảng điều khiển giúp bỏ qua quá trình lập trình máy tính thông qua các thiết bị phần cứng.*

Khái niệm mới

Lập trình máy tính bằng cách bật (On) và tắt (Off) được gọi là ngôn ngữ máy tính (machine language).

Mặc dù chương trình On/Off này không thể đọc trực tiếp, nhưng mỗi tổ hợp công tắc On/Off đưa ra lệnh xác định. Bằng cách kết hợp các lệnh, máy tính xử lý dữ liệu đặt ở vị trí nào đó trong bộ nhớ của máy tính. Chương trình gồm 30 lệnh phức tạp không phải làm bất cứ điều gì hơn ngoài việc thực hiện phép nhân hai số! Mặc dù có khó khăn trong việc lập trình, song phép tính nhanh như tia chớp như thế sẽ không là gì đối với máy tính. Máy tính được sử dụng trong quân đội hầu tính quỹ đạo đường đạn và các phép tính khác.

Mách nước

Qua tìm hiểu khái quát lịch sử lập trình, bạn sẽ đánh giá đúng những gì Visual Basic có thể thực hiện khi bắt đầu sử dụng Visual Basic cho các chương trình.

Thế giới máy tính đã bắt đầu tăng tốc bằng bảng điều khiển. Nhiều bộ nhớ đã được đưa vào, và chương trình có nhiều chức năng hơn. Cuối thập niên 40, người ta đã nảy ra sáng kiến thay thế bảng điều khiển bằng máy đánh chữ tương tự bàn phím. Thay thế tổ hợp On/Off, lập trình viên có thể nhập từ Add và Store trên bàn phím. Máy tính sẽ phân tích lệnh, tìm tổ hợp On/Off cần thiết và tự đặt các trạng thái điều khiển bên trong bộ nhớ.

Ngôn ngữ máy là vô cùng! Ngôn ngữ máy tồn tại mãi mãi! May mắn thay, bạn không phải viết chương trình trong bất kỳ ngôn ngữ máy On/Off nào. Ngôn ngữ máy tính ngày nay là sự gần gũi hơn của việc đàm thoại với máy qua các nút điều khiển On/Off. Ngay cả máy tính mạnh nhất hiện nay được lập trình bằng công cụ Visual Basic chẳng hạn, vẫn chỉ nhận biết ngôn ngữ máy. Mọi người không thích ngôn ngữ máy, bởi nó rất khó sử dụng. Máy tính không hề thích điều gì khác. Mọi ngôn ngữ lập trình đều đưa ra mã nguồn giúp bạn nhập vào ngôn ngữ lập trình và chuyển nó thành ngôn ngữ máy, sao cho máy tính có thể thi hành những lệnh đó.

QUÁ TRÌNH CẢI TIẾN NGÔN NGỮ LẬP TRÌNH

Một khi lập trình viên có bàn phím, thì không chỉ dừng lại ở đó. Ngôn ngữ phát triển từ hệ thống On/Off thành ngôn ngữ cấp cao hơn mà đọc vào cứ ngỡ là văn bản dùng để nói. Ví dụ 1.1 minh họa chương trình FORTRAN vắn tắt, một trong các ngôn ngữ cấp cao trước đây. Dù bạn chưa hiểu hết chương trình FORTRAN, song nó rất thiết thực trong lập trình, trái ngược với công tắc điều khiển On/Off, chương trình này tạo cơ hội cho nhiều người trở thành lập trình viên. Công nghệ phần mềm đã đơm hoa vào những năm 50, và các chương trình thiết kế phát triển từ công cụ tính toán đơn giản nhất, cho đến trình ứng dụng trong khoa học và thương mại.

Ghi chú

FORTRAN có nghĩa là trình dịch công thức (FORmula TRANslator). Nó được sử dụng chủ yếu trong lập trình khoa học và toán học. Dù cho FORTRAN là một trong các ngôn ngữ lập trình cấp cao cũ, nhưng hiện nay nhiều máy tính vẫn dùng chương trình FORTRAN. Phần lớn ngôn ngữ bắt nguồn từ ngôn ngữ FORTRAN trước đây.

Ví dụ 1.1. *Chương trình FORTRAN trước đây.*

```
WRITE (6, 10)
10 FORMAT('** Payroll Calculations **')
WRITE (6, 11)
11 FORMAT('** Enter the employee's ID, hours, and pay rate')
TAXRAT = 0.25
101 READ(5, 102, END=900) IDEMP, HRSWRK, RATE
102 FORMAT(I5, 1X, I3, F5.2)
IF (HRSWRK .GT. 40) GOTO 300
*******COMPUTE REGULAR PAY
GRSPAY = HRSWRK * RATE
GOTO 500
300 OVRHRS = HRSWRK - 40
*******COMPUTE OVERTIME PAY
OTGRS = OVRHRS * RATE *
1.5
```

```
GRSPAY = 40.0 * RATE + OTGRS
500 TAXES = GRSPAY * TAXRAT
PAYNET = GRSPAY - TAXES
WRITE (6,503) IDEMP, PAYNET
503 FORMAT('EMP: ', I3, 2X, 'NET PAY:', F6.2)
GOTO 101
*******END-OF-JOB PROCESSING
900 END
```

Nếu không quen với chương trình FORTRAN hay lập trình trong các ngôn ngữ khác, có lẽ bạn sẽ không hiểu hết Ví dụ 1.1. Tuy nhiên, bạn dễ dàng đoán một số từ là mã lệnh thực hiện chương trình tính lương. Thực ra, FORTRAN không cung cấp mã lệnh có khả năng đọc hoàn hảo, nhưng các lệnh như IF, GOTO, và WRITE, nâng cao hiệu suất làm việc của lập trình viên so với các ngôn ngữ trước đó.

Với ngôn ngữ cấp cao như FORTRAN chẳng hạn, trong suốt hai thập niên 50 và 60 tốc độ phát triển phần mềm thật đáng nể. Việc lập trình trở nên gần gũi với nhiều người vì ngôn ngữ lập trình dễ sử dụng hơn. Số lượng chương trình gia tăng khiến máy tính bán rất chạy. Công ty, thư viện và trường học đều tìm mua máy tính. Không có thế hệ máy tính nào trước đó lại có nhiều chức năng được khai thác đến như vậy. Khi các công ty máy tính bán chạy, sự cạnh tranh làm giảm giá thành và sản xuất nhiều máy tính mạnh hơn. Và như bạn đã thấy những gì diễn ra trong thập niên 90, thật là kỳ diệu.

Khái niệm mới

Syntax – Cú pháp, nghĩa là cách viết và văn phạm của ngôn ngữ.

Dù ngôn ngữ lập trình rất dễ học song vẫn cần những phương thức lập trình trợ giúp.. Một số giáo sư trường đại học Dartmouth quyết định viết ngôn ngữ lập trình đơn giản hơn dựa trên FORTRAN nhưng đòi hỏi ít chi tiết hơn FORTRAN. Những giáo sư này đã thiết kế ngôn ngữ BASIC, vốn là ngôn ngữ lập trình cho phép sinh viên viết chương trình dễ dàng và ít bị ràng buộc về mặt cú pháp hơn FORTRAN.

Ghi chú

BASIC là chữ viết tắt của Beginner's All–purpose Symbolic Instruction Code. Nghĩa đầy đủ của nó dài dòng và khó nhớ hơn nhiều bản thân ngôn ngữ BASIC.

Ví dụ 1.2 chỉ ra nét tương đồng giữa chương trình BASIC với chương trình FORTRAN trong Ví dụ 1.1. Với người mới lập trình, ví dụ chương trình BASIC này có lẽ không dễ chút nào so với mã lệnh FORTRAN. Tuy nhiên, ngôn ngữ BASIC rõ ràng và dễ hiểu hơn ngôn ngữ tiền nhiệm FORTRAN. Mặc dù công việc lập trình vẫn phải nỗ lực, song BASIC đã mang lại một số khởi sắc cho việc lập trình và cuốn hút nhiều người tìm đến công việc lập trình hơn.

Ví dụ 1.2. *Chương trình BASIC.*

```
10 PRINT "** Payroll Calculations **"
20 PRINT
"** Enter the employee's ID, hours, and pay rate"
30 TAXRAT = .25
40 INPUT IDEMP$, HRSWRK, RATE
50 IF (IDEMP$ = "END") THEN GOTO 160
60 IF (HRSWRK > 40) GOTO 70
70 REM *******COMPUTE REGULAR PAY
80 GRSPAY = HRSWRK * RATE
90
GOTO 110
100 OVRHRS = HRSWRK – 40
110 REM *******COMPUTE OVERTIME PAY
120 OTGRS = OVRHRS * RATE * 1.5
130 GRSPAY = 40 * RATE + OTGRS
140 TAXES = GRSPAY * TAXRAT
150 PAYNET = GRSPAY – TAXES
160 PRINT "EMP: "; IDEMP$; "NET PAY:";
PAYNET
170 GOTO 20
180 END
```

Đến thập niên 80, số lượng lập trình viên tiếp tục gia tăng. Mặc dù có nhiều lập trình viên mới, song bản thân công cụ lập trình vẫn chưa được cải tiến. Nhiều người thiết kế ngôn ngữ lập trình mới, tuy gọi là "nâng cao và cải tiến", nhưng phần lớn ngôn ngữ vẫn giữ đặc tính thủ tục nguyên bản mà FORTRAN và BASIC đã cung cấp.

Quá trình cải tiến toàn diện công cụ lập trình không diễn ra cho đến khi thay đổi phần cứng: NASA đã nỗ lực thiết kế *mạch tổ hợp nhỏ* (microchip) và cho ra đời máy vi tính. Giờ đây, thay cho các lập trình viên mới có khả năng, máy tính để bàn tạo cơ hội cho hàng triệu người trở thành lập trình viên. Lập trình viên cần công cụ lập trình dễ dàng hơn.

Khái niệm mới

***QBasic** là bản sao ngôn ngữ BASIC gần đây.*

Mọi nỗ lực cải tiến đã làm cho ngôn ngữ lập trình như BASIC ngày càng thông dụng. Phiên bản mới của BASIC, ví dụ như QBasic, cho phép lập trình viên viết chương trình hoàn chỉnh nhưng tốn ít công sức. Lập trình viên cũng có thể viết chương trình linh hoạt hơn trước đây. Ví dụ 1.3 trình bày phiên bản QBasic của chương trình tính lương tương tự hai ví dụ trước. Như bạn có thể thấy, ngôn ngữ ít khắt khe và tự do hơn.

Ví dụ 1.3. *Chương trình Qbasic.*

```
TaxRate = .25
PRINT "** Payroll Calculations **"
PRINT "** Enter the employee's ID, hours, and pay rate"
INPUT IdOfEmp$, HrsWorked, Rate
DO UNTIL (IdOfEmp$ = "END")
 IF (HrsWorked <= 40) THEN
 '
*******Compute regular pay
 GrossPay = HrsWorked * Rate
 ELSE
 ' *******Compute overtime pay
 OverTimeHours = HrsWorked - 40
 OverTimeGross = OverTimeHours * Rate * 1.5
```

```
GrossPay = 40 * Rate + OverTimeGross
END IF
Taxes = GrossPay * TaxRate
PayNet = GrossPay - Taxes
PRINT "Emp: "; IdOfEmp$; "Net Pay:"; PayNet
INPUT IdOfEmp$, HrsWorked, Rate
LOOP
END
```

CHẠY CHƯƠNG TRÌNH ĐỂ XUẤT KẾT QUẢ

Khái niệm mới

Máy tính đưa ra kết quả trên màn hình và máy in.

Một công việc chỉ có kết quả khi các lệnh trong chương trình xuất ra một kết quả quan trọng. Bằng nỗ lực lập trình, chương trình hoàn thành chỉ thị máy tính trực tiếp thi hành tác vụ, như tính lương chẳng hạn. Sau khi nhập chương trình hay nạp chương trình từ đĩa, bạn phải chỉ thị máy tính chạy hoặc thi hành chương trình. Sau đó, máy tính thực hiện lệnh của chương trình và xuất kết quả. Nếu bạn nhập mã lệnh ở Ví dụ 1.3 vào ngôn ngữ lập trình QBasic, kết quả sẽ như sau:

```
** Payroll Calculations **
** Enter the employee's ID, hours, and pay rate
? KL823, 40, 9.25
Emp: KL823 Net Pay: 277.5
? PO341, 35, 10
Emp: PO341 Net Pay: 262.5
? LP543, 40, 10
Emp: LP543 Net Pay: 300
? END, 0, 0
```

Ghi chú

Mã lệnh QBasic cho người dùng biết chương trình không xử lý thêm nhân viên nào nữa bằng cách nhập từ END thay thế mã số nhân viên và số 0 thay tổng số giờ công và bậc lương.

Mách nước

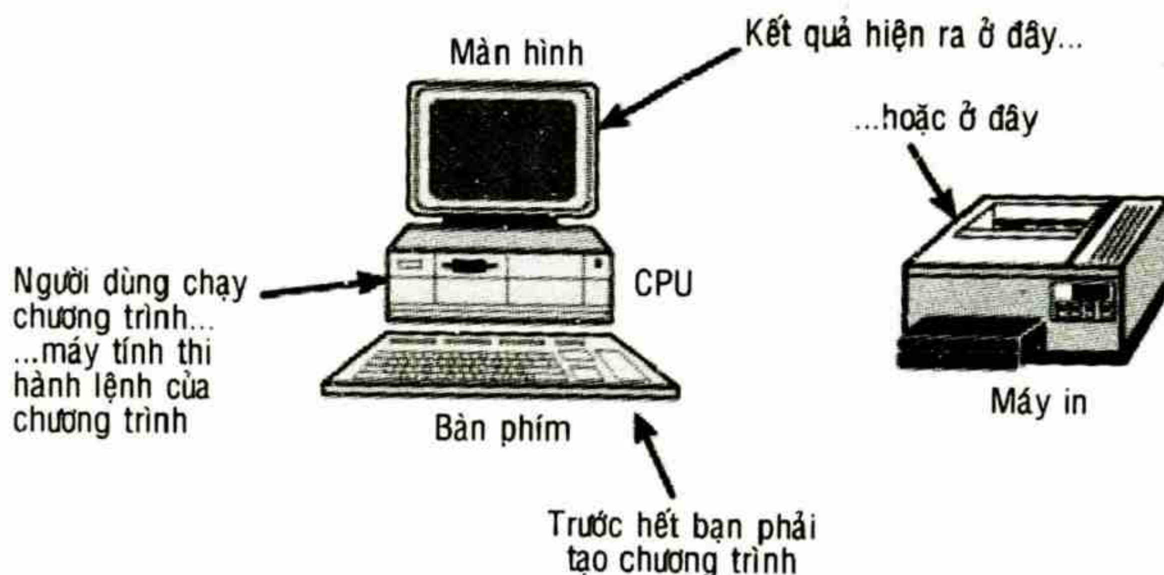
Kết quả xuất này sẽ khác đi nếu người dùng nhập vào dữ liệu hoàn toàn khác, bởi vì chương trình sẽ tính kết quả trả lương khác nhau.

Khái niệm mới

Input ***là dữ liệu người dùng nhập vào.***

Như bạn thấy trong lúc chạy chương trình, nó sẽ thực hiện tính tổng số tiền lương phải trả dựa trên dữ liệu nhập vào của người dùng. Dữ liệu có thể đến từ nhiều nguồn khác nhau. Chương trình có thể nhận dữ liệu nhập vào từ tập tin trên đĩa, modem kết nối qua đường dây điện thoại hoặc từ chương trình khác đang chạy song song với chương trình của bạn. Người dùng không phải khi nào cũng thấy được kết quả xuất ra, chương trình thường thực hiện tính toán và tổ hợp dữ liệu, song gởi kết quả đến tập tin trên đĩa hoặc máy tính khác thông qua modem.

Hình 1.3 minh họa các bước từ khi viết chương trình cho đến lúc xuất kết quả, như kết quả tính lương bạn đã thấy.



Hình 1.3. *Các bước từ lúc lập trình đến khi xuất kết quả.*

Chú ý

Kết quả xuất ra sau khi nhập bằng trình xử lý từ sẽ là văn bản hiển thị trên giấy hay trên màn hình. Kết quả xuất từ các chương trình khác nhau có thể đưa ra những bảng báo cáo lương, kết quả tính toán hoặc hình ảnh khác nhau. Lệnh của chương trình sẽ xác định những gì chương trình xuất ra.

KHÓ KHĂN TRONG QUÁ TRÌNH XỬ LÝ LỖI KỸ THUẬT

Khái niệm mới

Syntax – Cú pháp là văn phạm của ngôn ngữ.

Lỗi kỹ thuật (bug) trong chương trình máy tính làm cho lập trình viên cảm thấy khó chịu. Hiếm khi chương trình chạy ngay lần đầu tiên. Có hai loại lỗi: *lỗi cú pháp* và *lỗi logic*. Giả sử bạn chưa hiểu hết ngôn ngữ lập trình Visual Basic, những câu sau sẽ minh họa hai loại lỗi xuất hiện trong mã lệnh:

There are two errrors in this sentence.

What are the two errors? I'll wait...

Lỗi cú pháp rất dễ phát hiện. Từ *errrors* sai lỗi chính tả. Riêng lỗi logic phải mất nhiều thời gian tìm kiếm. Nội dung câu sai về mặt logic gọi là lỗi logic. Ví dụ ở câu đầu tiên, phải sửa lại là There is one error in this sentence, bởi vì chỉ có một lỗi cú pháp là errrors.

Khi bắt đầu viết chương trình, có lẽ bạn sẽ nhớ lại ví dụ này. Không mấy khó khăn trong quá trình tìm lỗi cú pháp, bởi vì Visual Basic tìm kiếm giúp bạn. Nói cách khác giả sử bạn nhập từ Tezt, trong lúc Visual Basic chỉ mong chờ từ Text, ngay lập tức Visual Basic hiển thị hộp báo lỗi giữa màn hình. Visual Basic rất bướng bỉnh khi gặp lỗi cú pháp, bởi lẽ nó không chấp nhận thi hành chương trình đến khi bạn xóa tất cả các lỗi cú pháp trong mã lệnh.

Tuy nhiên, khi bắt gặp lỗi logic, Visual Basic không thể giúp bạn. Ví dụ, trường hợp bạn đang viết thủ tục tính lương và vô ý trừ đi 50 đô–la trên mỗi phiếu chi trả của khách hàng. Visual Basic không có cách nào để nhắc không nên trừ con số đó. Còn như bạn chỉ thị Visual Basic trừ một số – dĩ nhiên bạn sử dụng cú pháp đúng – Visual Basic sẽ trừ đi con số đó. Chỉ đến khi khách hàng thắc mắc, bạn mới phát hiện lỗi logic.

Mách nước

Nên kiểm tra kỹ chương trình trước khi phân phối hay ứng dụng trong công việc quan trọng. Hãy chạy chương trình bằng cách dùng mọi tổ hợp dữ liệu hầu tìm kiếm và hiệu chỉnh càng nhiều lỗi logic càng tốt.

GIAO DIỆN ĐỒ HỌA THAY ĐỔI MỌI THỨ

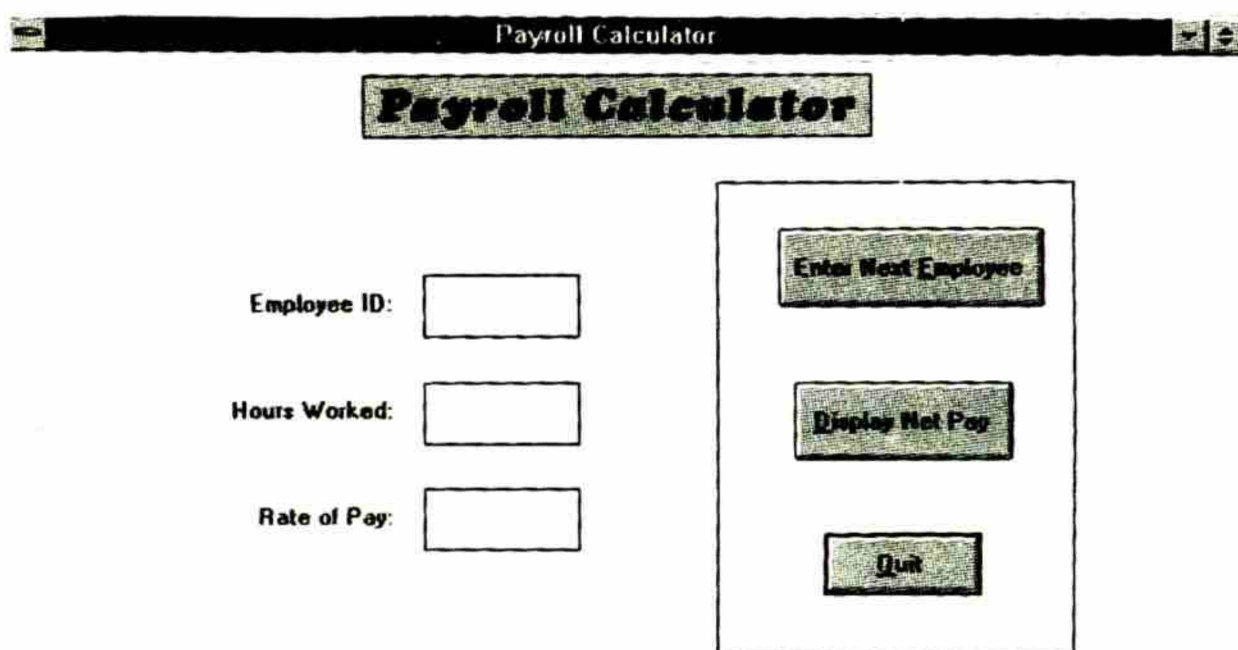
Khái niệm mới

GUI: *viết tắt của cụm từ Graphical User Interface – giao diện người dùng dạng đồ họa.*

Trong thập niên 80, lập trình gặp nhiều trở ngại khi môi trường trực quan GUI cấp cao như Microsoft Windows xuất hiện. Môi trường GUI đòi hỏi kết quả lập trình khó và cao cấp hơn môi trường tính toán dựa trên văn bản. Mặc dù người dùng và lập trình viên cần công cụ đơn giản hơn, song môi trường GUI lại đòi hỏi nhiều công cụ lập trình phức tạp.

Khái niệm mới

Event – biến cố có thể là lần nhấp mouse, nhấn phím, chọn menu hoặc hoạt động Windows ở bên trong.



Hình 1.4. *Người dùng điều khiển tùy ý chương trình hướng biến cố.*

Quá trình xuất kết quả dựa trên văn bản như đã nêu ở phần trước tương đối dễ. Để xuất cửa sổ – như màn hình minh họa ở Hình 1.4 – không đơn giản chút nào. Người dùng không phải nhập vào mã số ID, số ngày công và số tiền theo thứ tự thiết lập. Thậm chí người dùng

có thể nhấn ba nút lệnh bên phải màn hình theo thứ tự tùy ý. Trong chương trình Windows, biến cố đặc trưng xác định tiến trình hành động tiếp theo. Chương trình Windows được gọi là chương trình hướng biến cố, trái ngược với chương trình thủ tục dựa trên văn bản. Trong Windows, người dùng định những gì xảy ra kế tiếp bằng cách kích hoạt biến cố mà chương trình Visual Basic sẽ đáp ứng.

Củng cố

Lập trình máy tính đã có tiến bộ vượt bậc khi bảng điều khiển thay thế các linh kiện điện tử trước đây. Đến khi bàn phím ra đời, lập trình viên có thể thiết kế nhanh hơn. Cho đến Visual Basic là bước nhảy vọt tiếp theo trong công cụ lập trình đã hiện hữu.

CHUYỂN TIẾP TỪ BASIC SANG VISUÀL BASIC

Khái niệm

Khi thiết kế công cụ lập trình Windows, tính phức tạp dường như gia tăng. Môi trường Microsoft Windows cần cho công việc lập trình phức tạp hơn chương trình không thuộc môi trường Windows dựa trên DOS. Visual Basic được thiết kế hầu cung cấp phương thức đơn giản để viết chương trình Windows trong khi vẫn giữ lại tất cả các lệnh của ngôn ngữ lập trình QBasic.

Định nghĩa

Chương trình dựa trên nền văn bản thường gọi là chương trình thủ tục.

Ngôn ngữ thủ tục như FORTRAN và QBasic đơn giản không năng động trong môi trường GUI. Thực ra, chủ yếu là do người dùng tự nâng cao vai trò của GUI là xử lý hướng biến cố như đã giải thích ở mục trước. Người dùng thường không thực hiện các thao tác theo thứ tự y hệt nhau. Nghĩa là, người dùng có thể chọn menu xổ xuống, nhấp mouse, nhập văn bản, hay chọn lựa câu trả lời đúng, họ có thể thực hiện theo thứ tự tùy ý. Ngôn ngữ lập trình thủ tục yêu cầu chương trình điều khiển thứ tự hành động của người dùng. Windows đòi hỏi cách điều khiển khác. Ngôn ngữ lập trình truyền thống không thể xử lý GUI hoàn hảo.

Có thể viết chương trình hướng biến cố trong ngôn ngữ thủ tục. Hiện nay đa số chương trình đang sử dụng trong Windows được viết bằng ngôn ngữ C, vốn là ngôn ngữ thủ tục giống với FORTRAN hơn là Visual Basic, mặc dù C và FORTRAN rất khác nhau về mặt cú pháp và cách tiếp cận. Tuy nhiên, lập trình viên gặp nhiều khó khăn khi sử dụng chương trình C hơn là Windows bởi lẽ C là ngôn ngữ thủ tục trong khi Windows là ngôn ngữ hướng biến cố.

Nhiều giải pháp lập trình được đưa ra nhằm trợ giúp các lập trình viên Windows. Khái niệm *lập trình hướng đối tượng* (OOP) tương đối mới giúp lập trình Windows gặp thuận lợi hơn – giống lập trình hướng biến cố hơn là các ngôn ngữ như FORTRAN, BASIC, hay C. Tuy nhiên, ngay cả OOP, cũng làm cho lập trình viên phải đau đầu vì các bộ phận lập trình luôn bận rộn và bị ùn tắc mà không có hướng xử lý.

Vấn đề quan trọng là khi máy tính trở nên dễ dàng hơn đối với người dùng bằng cách cung cấp môi trường đồ họa như Windows chẳng hạn, thì chương trình mà người dùng sử dụng trở nên khó thiết kế hơn. Vì vậy, nhu cầu cung cấp chương trình GUI gia tăng làm cho các đơn đặt hàng viết chương trình trở nên quá tải. Microsoft đã giới thiệu ngôn ngữ lập trình mới cách đây vài năm chính là Visual Basic. Sau đây là những điểm thuận lợi của Visual Basic:

- Visual Basic được thiết kế dựa trên ngôn ngữ lập trình QBasic, vì vậy lập trình viên cảm thấy Visual Basic quen thuộc hơn các ngôn ngữ lập trình khác.
- Thiết kế Visual Basic thành ngôn ngữ lập trình Windows. Quá trình thiết kế Visual Basic cơ bản gắn liền với khái niệm lập trình hướng biến cố. Thực ra, chương trình thủ tục khó viết bằng Visual Basic.
- Visual Basic là ngôn ngữ lập trình đồ họa theo hướng tiếp cận. Thật vậy, bạn có thể tạo một chương trình Visual Basic chạy thông suốt bằng cách di chuyển biểu tượng bật tắt trên màn hình mà không cần viết bất kỳ lệnh nào bằng ngôn ngữ lập trình Visual Basic.

Visual Basic là chương trình tương tự màn hình xuất kết quả. Nói cách khác, để thiết kế và viết chương trình đơn giản như Ví dụ 1.3, bạn sẽ đặt văn bản, nút và kẻ các đường trên màn hình bằng cách sử dụng

công cụ trực quan được cung cấp trong Visual Basic đến khi chương trình y hệt chương trình Windows mong muốn. Việc sắp xếp thành phần trực quan sẽ tạo trang lệnh thay thế trang lệnh được nhập khi sử dụng ngôn ngữ lập trình thủ tục truyền thống.

Ghi chú

Sau khi thành công với Visual Basic for Windows, Microsoft thiết kế Visual Basic for DOS. Tuy nhiên, Visual Basic for DOS chưa mấy hoàn hảo và ngày càng bị mai một.

Trước khi học lập trình bằng Visual Basic, bạn phải am hiểu các công cụ của nó. Bài 2 sẽ giới thiệu về Visual Basic. Bạn sẽ học cách cài đặt, khởi động và sử dụng màn hình Visual Basic. Một khi nắm vững các cơ chế về Visual Basic, bài học tiếp sẽ chỉ dẫn từng bước lập trình thông qua quá trình làm quen với thiết kế và thi hành trình ứng dụng Visual Basic đầu tiên.

Visual Basic là ngôn ngữ hướng biến cố và thủ tục. Không nên nghĩ rằng cơ chế viết lệnh chương trình là lạc hậu. Thực tế, một trong những thành phần quan trọng nhất của Visual Basic là ngôn ngữ thủ tục BASIC, giống như ngôn ngữ lập trình trong môi trường trực quan. Khi bắt đầu mở rộng lập trình Visual Basic bằng thiết kế nhiều chương trình hoàn chỉnh hơn, bạn cần phối hợp thành phần trực quan của ngôn ngữ với thủ tục QBasic.

Chú ý

Bài học kết thúc ở đây mà không mô tả chi tiết về Visual Basic bởi vì phần còn lại của cuốn sách sẽ đề cập cụ thể. Bây giờ, bạn đã có một cái nhìn cơ bản về lập trình và những khó khăn trong lập trình Windows, hãy sẵn sàng để bắt đầu các bài học bổ ích về Visual Basic. Sau khi học xong 24 bài học, bạn sẽ làm chủ Visual Basic!

Củng cố

Trong nhiều phương pháp, viết chương trình là phương pháp cũ nhất – nghĩa là, quá trình viết trang lệnh văn bản cho máy tính thi hành gặp nhiều khó khăn và mất thời gian. Bắt đầu từ lệnh đầu tiên và hãy cẩn thận để không bỏ sót bước nào. Danh sách lệnh sẽ quyết định tất cả, và mặc dù có bỏ qua một hay hai bước, các lệnh cũng phải được thi hành tuần tự nếu muốn chương trình hoàn thành và đưa ra kết quả.

Chương trình viết bằng Visual Basic dễ sử dụng hơn bởi lẽ Visual Basic vẫn là một trong những công cụ dễ và nhanh nhất mà bạn có thể tìm thấy nhằm thiết kế trình ứng dụng Windows.

Ghi chú

Hầu hết các mục thường kết thúc bằng phần củng cố giúp bạn ôn lại chủ đề trong mục đó kèm theo bài thực hành lập trình. Mục đích chính của bài đầu tiên này cho bạn cái nhìn tổng quan về lịch sử lập trình. Vì vậy, không có bài tập thực hành trong bài này.

Bài tập

Kiến thức tổng quát

1. Chương trình là gì?
2. Nếu bạn cố sử dụng máy tính mà không có chương trình nào đang chạy thì kết quả như thế nào?
3. Hai cách để nhận được chương trình trên máy tính.
4. Quá trình viết chương trình gặp thuận lợi gì?
5. Điểm bất lợi trong lúc viết chương trình.
6. Đúng hay Sai: Một số công ty bán các chương trình mà bạn có thể chỉnh sửa cho phù hợp với nhu cầu của mình.
7. Mã lệnh là gì?
8. Thế nào là lỗi kỹ thuật?
9. Cho biết hai loại lỗi thường xảy ra trong khi viết chương trình.
10. Nếu viết sai lệnh Visual Basic, bạn đã vi phạm loại lỗi nào?
11. Visual Basic tìm cho bạn những loại lỗi nào?
12. Loại lỗi nào khó tìm nhất?
13. Thế hệ máy tính đầu tiên lập trình như thế nào?
14. Đưa thêm các công tắc chuyển đổi vào máy tính trước đây đã nâng cao khả năng lập trình của máy tính này như thế nào?
15. Ngôn ngữ lập trình đầu tiên, được gọi là ngôn ngữ máy tính, bao gồm tổ hợp trạng thái On/Off do bàn công tắc chuyển đổi cung cấp, tổ hợp On/Off mô tả điều gì?

16. Bộ phận phần cứng nào được bổ sung khiến nhiều người truy xuất và viết chương trình cho máy tính nhiều hơn?
17. Mã lệnh do lập trình viên nhập gọi là gì?
18. Tối thiểu thì mỗi ngôn ngữ lập trình cấp cao hiểu điều gì trước khi máy tính có thể hiểu ngôn ngữ?
19. Tên của một trong những ngôn ngữ lập trình cấp cao sớm nhất mà ngày nay vẫn được sử dụng trong lập trình khoa học và toán học là gì?
20. Ngôn ngữ nào được số giáo sư trường đại học Dartmouth thiết kế hầu khắc phục những trở ngại do ngôn ngữ lập trình đem lại?
21. BASIC thay thế cái gì?
22. Loại ngôn ngữ lập trình nào cần thiết trong môi trường DOS?
23. Loại ngôn ngữ lập trình nào cần dùng trong môi trường Windows?
24. Ý nghĩa của GUI là gì?
25. Cho hai ví dụ về biến cố.
26. Những thách thức phải vượt qua của chương trình hướng biến cố là gì?
27. Đúng hay Sai: Visual Basic không chứa bất kỳ công cụ lập trình thủ tục nào vì nó không có ích trong môi trường GUI.
28. Đúng hay Sai: Chương trình Visual Basic thường giống với dữ liệu xuất.

Phần nâng cao

29. Lập trình viên không ngồi trước máy tính nhập chương trình đã hoàn tất – Visual Basic đảm nhận công việc dễ dàng đó. Hãy mô tả các bước cần thiết để thiết kế chương trình.
30. Tại sao các lập trình viên nên kiểm tra toàn bộ chương trình họ đã viết?

Bài 2
Tổng quan về Visual Basic

- Cài đặt Visual Basic
- Khởi động và thoát khỏi Visual Basic
- Môi trường làm việc của Visual Basic
- Năm kiểu cửa sổ
- Màn hình Visual Basic
  - Thanh menu
  - Các phím truy xuất nhanh
  - Thanh công cụ: nhấp nhanh nút
- Bộ chỉ dẫn đơn vị đo

Bài này giúp bạn cài đặt hệ thống lập trình Visual Basic. Quá trình cài đặt Visual Basic rất dễ bởi Windows thực hiện hầu hết mọi việc thay bạn. Nếu đã từng cài đặt các chương trình Windows khác, bạn không phải lo lắng gì về việc cài đặt Visual Basic.

Chú ý

Tập sách giả thiết bạn đã từng sử dụng Windows. Cho dù không phải là chuyên gia về chương trình này, nhưng bạn nên làm quen với việc khởi động Windows, dùng mouse, chọn lệnh đơn trong menu Windows .v.v.

CÀI ĐẶT VISUAL BASIC

Khái niệm

Hãng Microsoft đã phát hành nhiều phiên bản Visual Basic. Khi phiên bản 1.0 tung ra thị trường, Visual Basic làm thay đổi cách suy nghĩ của mọi người về lập trình trên Windows. Ngay cả phiên bản đầu tiên, Visual Basic đã có công cụ lập trình hùng mạnh để giới thiệu đến lập trình viên Windows. Với cách sắp xếp chương trình và thành phần trên màn hình trực quan của Visual Basic, bạn sẽ vẽ toàn bộ chương trình thay vì viết chúng. Với từng phiên bản Visual Basic, Microsoft đã thêm vào nhiều hàm và khả năng lập trình.

Bạn cần một phiên bản hệ thống Visual Basic để học về Visual Basic và tạo các chương trình trên Windows trong tập sách này. Visual Basic 6.0 có hai phiên bản chính là Standard / Professional Edition và Enterprise Edition. Các ví dụ trong sách sử dụng phiên bản Visual Basic 6.0 – Enterprise Edition. Tuy nhiên, bạn cũng có thể sử dụng các phiên bản khác.

Phiên bản Visual Basic 6.0 Enterprise Edition bao gồm tất cả công cụ cần khai thác Visual Basic. Bạn sẽ đưa những điều khiển nâng cao vào chương trình khi cần và có thể thiết kế các chương trình xử lý y hệt trình ứng dụng Windows mà bạn sử dụng hàng ngày.

Thuật ngữ mới

Trình biên dịch tạo chương trình độc lập.

Nếu bản Visual Basic của bạn chưa có trình biên dịch, bạn có thể mua bản gốc, tuy phải tốn nhiều tiền hơn, ngược lại bạn có thể tạo chương trình thi hành độc lập. Bản gốc Visual Basic cho phép bạn tạo chương trình giống hệt các bản Visual Basic không có trình biên dịch. Nói cách khác, nếu bạn dùng một phiên bản không có trình biên dịch, khi muốn chạy chương trình bạn viết, trước hết bạn phải chạy Visual Basic và sử dụng lệnh trên menu Visual Basic để tải và chạy chương trình.

Các bước cài đặt phiên bản Visual Basic 6.0 – Enterprise Edition trong bộ Microsoft Visual Studio 6.0 trên hệ thống như sau:

1. Khởi động Windows.
2. Chèn đĩa CD–ROM có phiên bản Visual Basic 6.0 cần cài đặt vào ổ đĩa CD của bạn.
3. Từ màn hình Windows, nhấp đúp biểu tượng My Computer.
4. Nhấp đúp biểu tượng ổ đĩa CD–ROM của bạn.
5. Nhấp đúp biểu tượng Setup.exe để chạy chương trình cài đặt.
6. Bạn sẽ trả lời các câu hỏi của chương trình setup, cài các thành phần phụ, sau đó chọn ô chọn Microsoft Visual Basic 6.0, trước khi chương trình setup tự động cài đặt chương trình Visual Basic 6.0 cho bạn. Chương trình sẽ tự động cài biểu tượng chương trình Microsoft Visual Basic 6.0 trong nhóm chương trình Visual Studio 6.0 mới tạo.

7. Khi việc cài đặt hoàn tất, bạn sẽ chọn hoặc khởi động lại máy tính hoặc trở về Windows. Còn bây giờ, hãy quay về Windows để có thể tìm hiểu cách thức khởi động chuẩn.

Khái niệm mới

Default là giá trị mặc định được sử dụng khi bạn chưa nhập giá trị khác.

Cảnh báo

Tùy thuộc vào quá trình cài đặt Windows, một hay hai hộp thông báo sẽ xuất hiện hầu cho bạn biết tập tin đặc biệt mà chương trình cài đặt cần phải viết chồng. Bạn có thể nhấn nút Yes để tiếp nhận thông tin và tiếp tục.

Khái niệm mới

Nút điều khiển (control button) xuất hiện ở góc trái trên mỗi cửa sổ.

Củng cố

Máy tính của bạn giờ đã cài đặt chương trình Visual Basic. Hãy sẵn sàng khởi động chương trình và điều khiển môi trường.

KHỞI ĐỘNG VÀ THOÁT KHỎI VISUAL BASIC

Khái niệm

Muốn viết chương trình Windows bằng Visual Basic, bạn phải khởi động Visual Basic. Trước khi thoát Windows, bạn buộc phải thoát Visual Basic.

Khởi động Visual Basic

Bạn có thể khởi động Visual Basic trong Windows bằng cách nhấp menu Start, chọn menu con Microsoft Visual Studio 6.0, sau đó nhấp mục Microsoft Visual Basic 6.0, chương trình Visual Basic sẽ được nạp và xuất hiện trên màn hình.

Lưu ý

Tùy vào phiên bản Visual Basic bạn đã cài đặt, các bước cài đặt cũng như cách khởi động chương trình có thể khác đi đôi chút.

Một hộp thoại tương tự như Hình 2.1 sẽ xuất hiện ngay khi bạn khởi động Visual Basic.

New Project

Microsoft Visual Basic

New | Existing | Recent

Standard EXE · VBAPWIZ · WIZMAN · ADDIN · AXDOCDLL

AXDOCEXE · DATAPROJ · DHTMLAPP · IISAPP · VBENTCTL

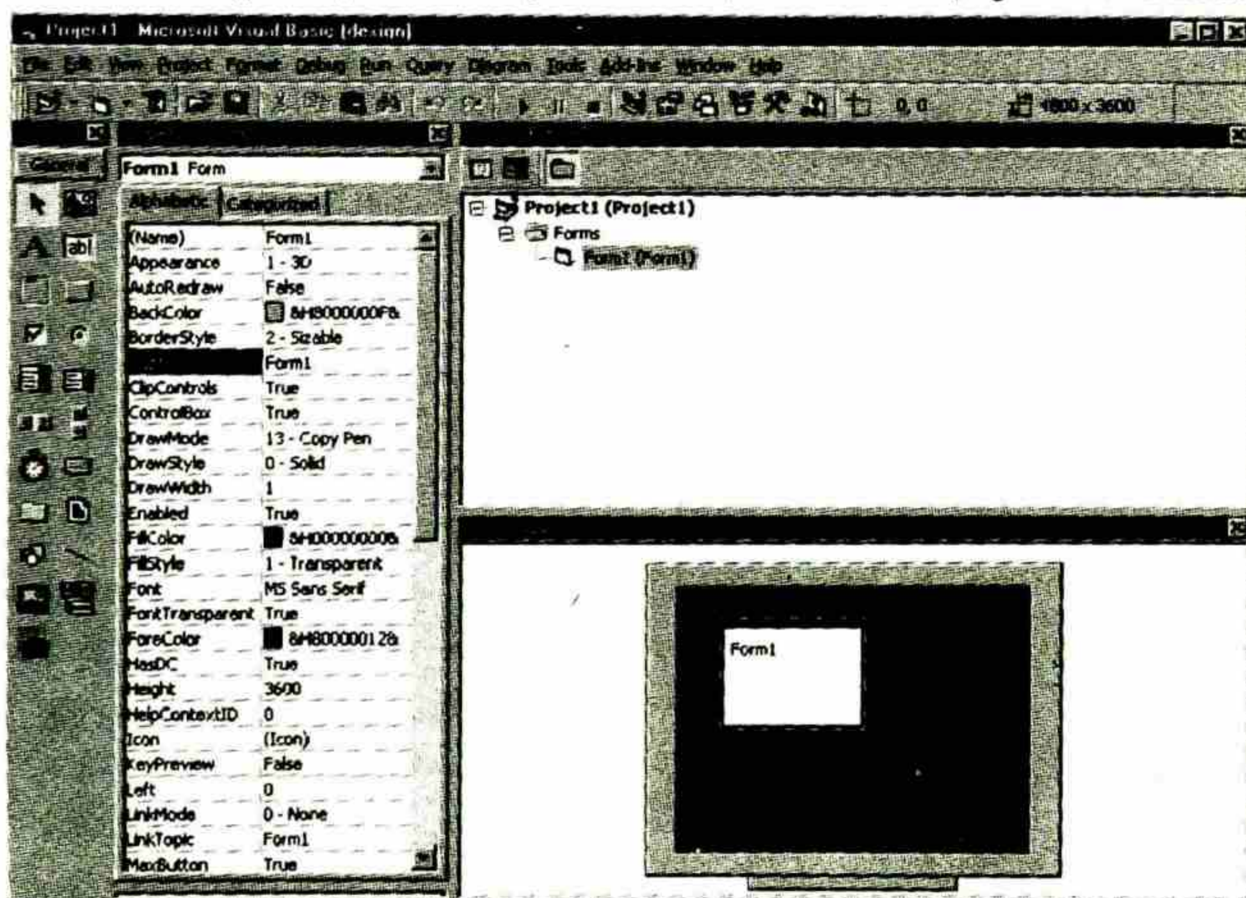
Open

Cancel

Help

Don't show this dialog in the future

Hình 2.1. *Hộp thoại New Project xuất hiện khi khởi động Visual Basic.*



Hình 2.2. *Màn hình Visual Basic mới đầu trông có vẻ khó hiểu.*

Bạn có thể điều chỉnh kích cỡ sắp xếp lại hay giấu bớt các thành phần trong màn hình Visual Basic, khi đó màn hình của bạn có thể trông khác Hình 2.2.

Thoát khỏi Visual Basic

Bạn sẽ thoát khỏi Visual Basic và trở lại Windows giống như cách bạn thoát nhiều trình ứng dụng Windows: chọn File ➡ Exit; nhấp nút Close trên cửa sổ chính của chương trình Visual Basic; nhấn tổ hợp phím Alt+F4; hoặc nhấp đúp biểu tượng Menu điều khiển của Visual Basic ở góc trái trên màn hình.

Nếu bạn đã thực hiện thay đổi một hay nhiều tập tin trong Project đang mở hiện hành (hãy nhớ rằng Project là tập hợp các tập tin liên quan đến trình ứng dụng của bạn), Visual Basic sẽ cho bạn cơ hội cuối cùng để lưu các thay đổi đó trước khi trở về Windows.

Lưu ý

Luôn thoát khỏi Visual Basic trước khi tắt máy tính nếu không bạn sẽ mất một phần hay toàn bộ chương trình đang viết.

Củng cố

Để khởi động Visual Basic chỉ cần nhấp đúp biểu tượng Microsoft Visual Basic 6.0 bên trong nhóm chương trình Microsoft Visual Studio 6.0. Thoát khỏi Visual Basic và quay lại Windows, bạn có thể chọn File ➡ Exit, hoặc nhấn tổ hợp phím Alt+F4, hay nhấp nút Close trong cửa sổ chính của Visual Basic.

MÔI TRƯỜNG LÀM VIỆC CỦA VISUAL BASIC

Khái niệm

Trước khi học quy cách viết chương trình bằng Visual Basic, bạn phải tìm hiểu mọi thứ trên màn hình Visual Basic. Một số người cho rằng màn hình Visual Basic khó hiểu lúc mới làm quen. Màn hình Visual Basic không chỉ khó hiểu lúc mới làm quen mà bạn còn phải tốn nhiều thời gian mới sử dụng được nó! Thực ra, màn hình Visual Basic lộn xộn hơn là khó hiểu. Một khi đã biết cách thức quản lý các thành phần trên màn hình, bạn sẽ cảm thấy dùng Visual Basic dễ dàng hơn.

Lập trình viên Visual Basic làm việc hiệu quả nghĩa là biết cách tổ chức lại màn hình Visual Basic khi cần. Visual Basic thật ra không giống chương trình Windows khác. Ví dụ, Microsoft Word là một trong các chương trình Windows hoàn hảo nhất và được sử dụng nhiều nhất. Khi sử dụng Word, về cơ bản bạn làm việc trong cửa sổ khổng lồ. Bạn có thể mở cửa sổ tài liệu thứ hai và định lại kích thước của hai cửa sổ để thêm nhiều thành phần vào màn hình, nhưng phần lớn thời gian bạn làm việc trong một cửa sổ tài liệu đơn.

Trong Visual Basic, hầu hết thời gian bạn làm việc với những cửa sổ đang mở. Có nhiều cửa sổ – đôi khi gọi là cửa sổ khung mở cùng một lúc, và bạn luôn cần thông tin từ các cửa sổ này. Vì vậy, bạn nên làm quen với màn hình và các thành phần ngay từ bây giờ.

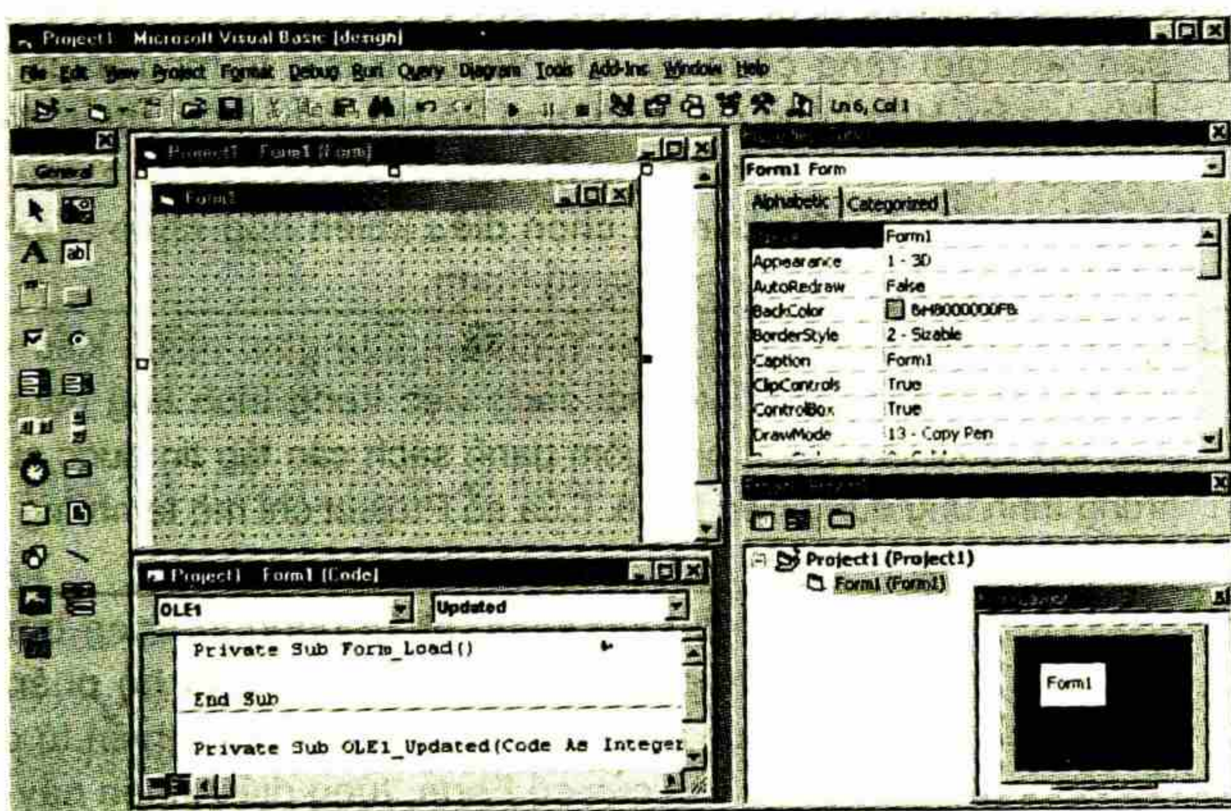
Củng cố

Hãy tìm hiểu thật kỹ lưỡng về môi trường Visual Basic để sau này bạn hiểu sâu hơn về ngôn ngữ và các điều khiển.

NĂM KIỂU CỬA SỔ

Khái niệm

Môi trường Visual Basic bao gồm nhiều kiểu cửa sổ mà bạn sẽ làm việc trong quá trình thiết kế trình ứng dụng.



Hình 2.3. *Các thành phần trên màn hình Visual Basic.*

Hình 2.3 minh họa thành phần chính của màn hình Visual Basic. Có lẽ bạn chưa hiểu hết tất cả các thành phần này trên màn hình, nhưng hãy học tên gọi của chúng để di chuyển đúng sau này khi bạn bắt đầu học cách lập trình Visual Basic.

Lưu ý

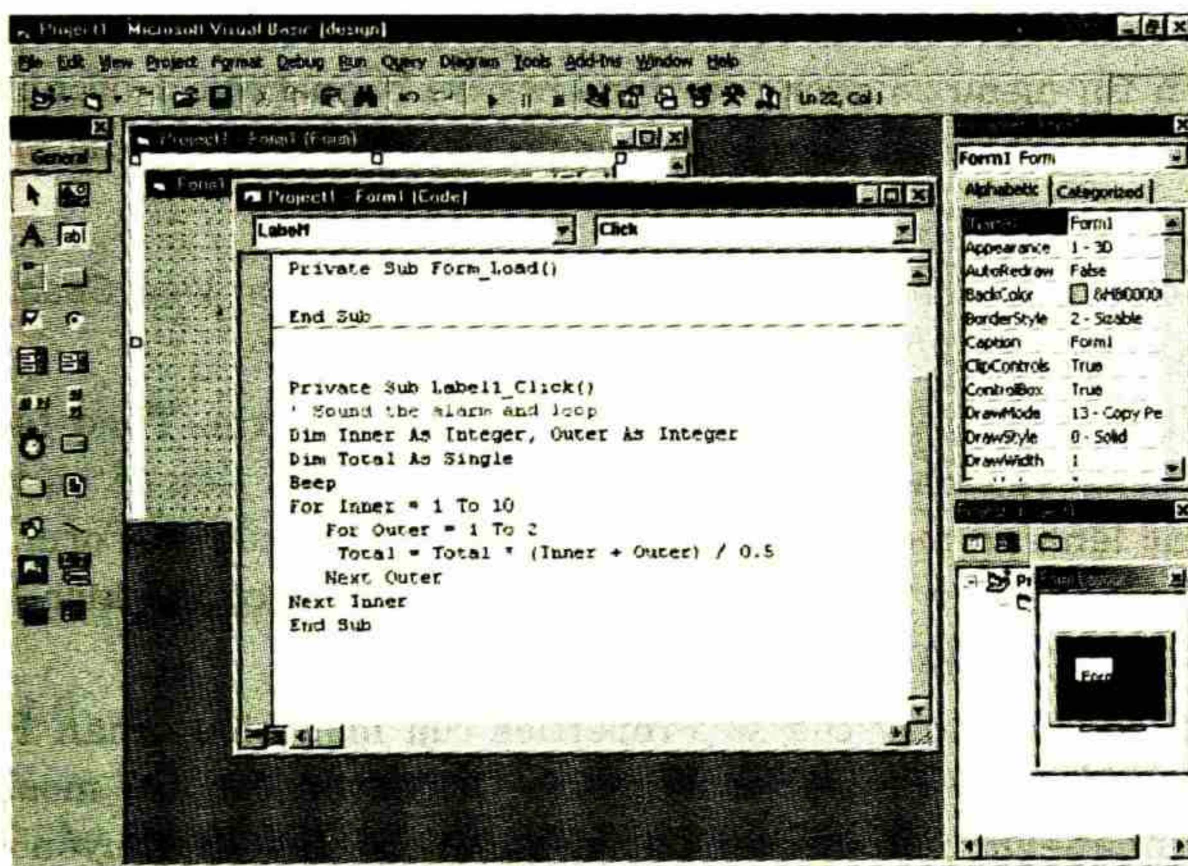
Thực ra, Hình 2.3 không phải là màn hình chính xác mà bạn có thể nhìn thấy khi khởi động Visual Basic lần đầu. Bạn sẽ học cách sắp xếp lại các thành phần màn hình cho giống với Hình 2.3. Hình 2.3 đã được tổ chức thích hợp để bạn dễ dàng nhìn thấy các thành phần chính trên màn hình.

Bảng 2.1 mô tả lần lượt năm kiểu cửa sổ chính của Visual Basic. Dù bạn không thể hiểu hết nội dung mô tả ngay lúc này, song hãy cố làm quen để không bỡ ngỡ khi học cách lập trình trong Visual Basic.

Bảng 2.1. *Các cửa sổ chính trong môi trường Visual Basic*

Cửa sổ	Mô tả
Form	Là địa điểm để bạn viết chương trình Windows. Bạn sẽ vẽ và đặt các khoản mục trên Form sao cho người dùng chương trình sẽ thấy và thao tác. Nếu bạn sử dụng trình xử lý từ trên Windows, cửa sổ sẽ chứa tài liệu vốn đang được hiệu chỉnh. Mặc dù không phải chương trình Visual Basic nào cũng đòi hỏi cửa sổ Form, nhưng đa số thì cần chúng để hiển thị thông tin và lấy thông tin từ người dùng.
Toolbox	Cửa sổ Toolbox chứa các công cụ. Điều này quá rõ ràng, nhưng bạn cần biết rằng công cụ của Visual Basic thường được gọi là điều khiển. Hộp công cụ là nơi bạn sẽ tìm thấy điều khiển đặt trên cửa sổ Form khi tạo chương trình Windows. Chẳng hạn, muốn hỏi người dùng một câu, bạn sẽ chọn điều khiển hộp nhập (Text Box) từ hộp công cụ và đặt vào cửa sổ Form.
Project	Chương trình Visual Basic trên Windows bao gồm nhiều kiểu và loại tập tin khác nhau phục vụ mọi công việc trong trình ứng dụng chạy độc lập. Cửa sổ Project bao gồm danh sách các tập tin được sử dụng trong trình ứng dụng hiện hành. Cửa sổ Project chỉ mô tả những tập tin chứa trên đĩa.
Properties	Cửa sổ Properties mô tả từng thành phần riêng lẻ trong trình ứng dụng. Ví dụ, cửa sổ Properties (thuộc tính) có một bộ phận chứa các thuộc tính như màu và kích thước. Khi đặt điều khiển từ cửa sổ Toolbox (hộp công cụ) vào cửa sổ Form, từng điều khiển này sẽ có

	một thuộc tính riêng. Một chương trình Visual Basic bất kỳ có thể có nhiều thành phần cùng với các thuộc tính, nhưng chỉ có một cửa sổ Properties. Trường hợp bạn muốn xem các thuộc tính khác nhau của một mẫu biểu hay điều khiển, thay đổi cửa sổ Properties nhằm hiển thị nhóm thuộc tính khác.
Code	Không giống với đa số các ngôn ngữ lập trình khác, bạn không phải viết nhiều mã lệnh khi thiết kế trình ứng dụng trong Visual Basic. Tuy nhiên, phần trực quan của Visual Basic sẽ loại trừ nhiều mã lệnh mà bạn phải viết nếu còn làm việc trong môi trường văn bản. Hình 2.4 minh họa cửa sổ Code (mã lệnh) chứa các thủ tục khá phức tạp. Mã lệnh trong cửa sổ Code là mã nguồn của chương trình (bạn đã tìm hiểu ở bài trước). Khi người dùng chạy chương trình, Visual Basic và máy tính biên dịch mã nguồn đó và thi hành lệnh trong mã nguồn.



Hình 2.4. *Cửa sổ Code với nhiều mã lệnh.*

Ghi chú

Mã lệnh trong cửa sổ Code chủ yếu thiết lập và cung cấp lệnh cho điều khiển mẫu biểu. Ví dụ, giả như bạn cần kiểm tra xem người dùng đã nhấp nút lệnh hay nhập câu trả lời chưa, có thể dùng mã lệnh để thực hiện tác vụ đó.

Như phần lớn cửa sổ đã được sử dụng trong các trình ứng dụng Windows, bạn có thể di chuyển, hiệu chỉnh kích thước, và đóng năm kiểu cửa sổ đã đề cập, đơn giản là dùng mouse thực hiện các tác vụ trên cửa sổ.

Khái niệm mới

Maximize ***nghĩa là phóng cửa sổ lên kích cỡ lớn nhất.***

Lấy ví dụ, khi bạn khởi động Visual Basic lần đầu tiên, cửa sổ Form che khuất các cửa sổ khác. Thông thường, Form là cửa sổ lớn nhất, bởi vì nó là cửa sổ nền của người dùng. Bạn có thể phóng lớn cửa sổ Form bằng cách nhấp đúp mouse trên thanh tiêu đề hay nhấp nút Maximize ở góc phải trên cửa sổ. Bây giờ hãy nhấp đúp thanh tiêu đề cửa sổ Form nhằm phóng lớn cửa sổ. Sau khi nhấp xong sẽ không còn cửa sổ nào bên trái màn hình.

Hiển nhiên, sẽ có cách xem menu và cửa sổ khác. Nhấn phím Alt hiện lại màn hình đầu để bạn có thể thấy menu và các thanh công cụ. Hãy hiển thị menu xổ xuống từ Windows, và chọn Project để nhìn thấy cửa sổ Project. Nhấp nút View Code trong cửa sổ Project để bật cửa sổ Code.

Mách nước

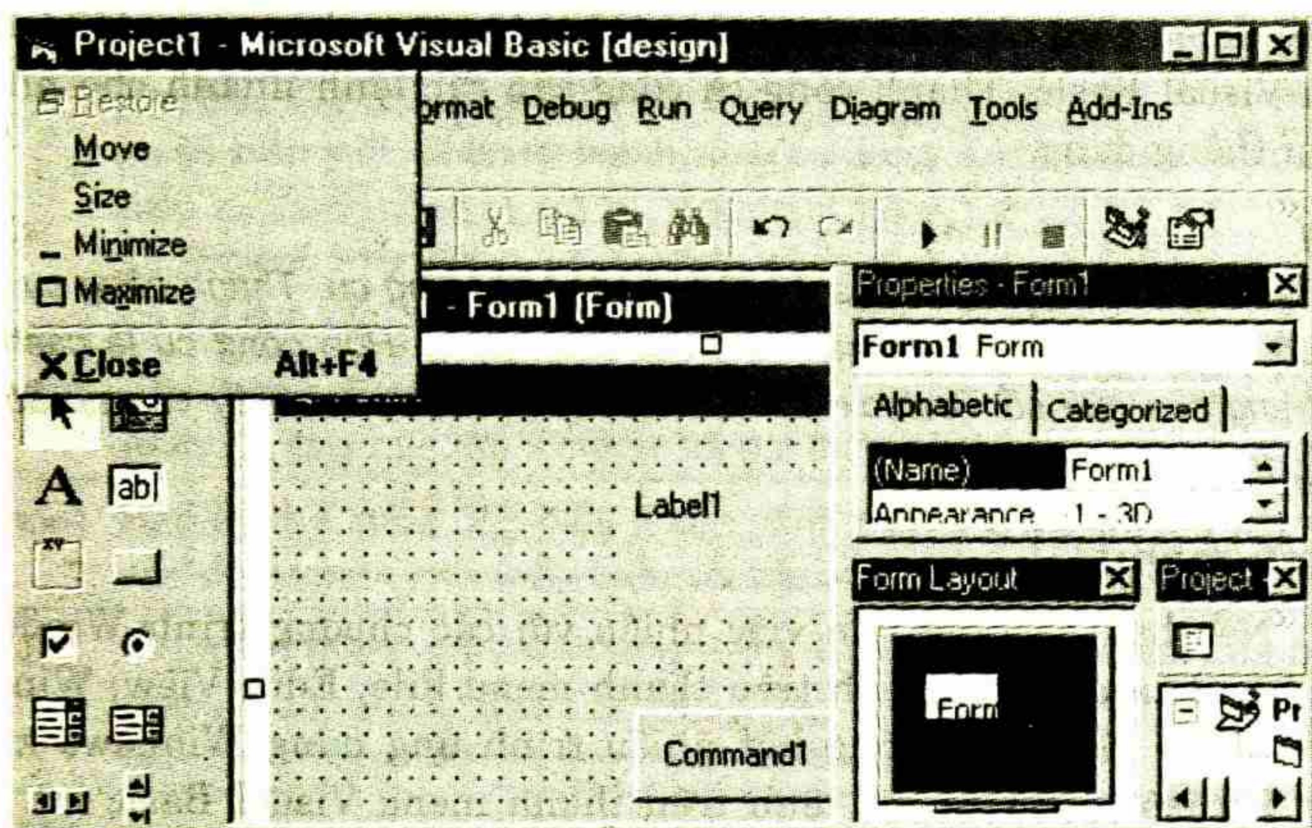
Có hai cách khác hiển thị cửa sổ Code. Có thể chọn View Code để xem cửa sổ. Cũng có thể nhấn F7 hiển thị mã lệnh.

Nhấp bất kỳ đâu trên vùng trống cửa sổ Form. Thao tác nhấp cửa sổ bất kỳ sẽ kích hoạt cửa sổ đó, bật sáng thanh tiêu đề và làm cho tất cả các lệnh và menu có hiệu lực. Khi kích hoạt cửa sổ Form, cửa sổ Project và Code sẽ ở đằng sau mẫu biểu, nhưng bạn có thể kích hoạt nó trở lại theo cách vừa được trình bày.

Muốn nhìn thấy cửa sổ Properties của mẫu biểu, nhấn F4 hoặc chọn lệnh View ➥ Properties Window. Di chuyển con trỏ mouse trên cửa sổ, nhấn–giữ nút mouse. Lúc này bạn có thể di chuyển cửa sổ Properties bằng cách kéo mouse trên màn hình. Khi bạn thả nút mouse, Visual Basic đặt cửa sổ tại điểm đó.

Hãy thử hiệu chỉnh kích thước cửa sổ Properties. Di chuyển con trỏ mouse đến góc tùy ý trên cửa sổ Properties. Biểu tượng mouse đổi thành hai mũi tên ngược chiều. Bằng cách nhấp và kéo mouse, bạn có thể định lại kích thước cửa sổ.

Khi đã sẵn sàng đóng cửa sổ, cách dễ nhất là nhấp đúp nút điều khiển. Tuy nhiên, nếu nhấp nút điều khiển một lần, bạn sẽ thấy menu điều khiển của cửa sổ, như Hình 2.5. Có lẽ bạn đã thấy menu điều khiển trong những tác vụ khác bạn đã thực hiện. Nếu không, bạn có thể sử dụng menu điều khiển để di chuyển, hiệu chỉnh kích cỡ và đóng cửa sổ. Tuy nhiên, dùng mouse như vừa mô tả dễ hơn nhiều so với menu điều khiển. Muốn đóng menu điều khiển, bạn có thể nhấp nút điều khiển một lần nữa hoặc nhấn phím ESC hai lần.



Hình 2.5. *Menu điều khiển.*

Củng cố

Năm kiểu cửa sổ chính của Visual Basic hỗ trợ quá trình định vị điều khiển và vùng làm việc mà bạn sử dụng để thiết kế trình ứng dụng Visual Basic. Cửa sổ Form là cửa sổ quan trọng nhất dành cho trình ứng dụng mà bạn sẽ viết bởi lẽ bạn sẽ vẽ và đặt các điều khiển tương tác cho người dùng làm việc trên cửa sổ Form. Các cửa sổ khác có sẵn hầu trợ giúp và cung cấp công cụ.

MÀN HÌNH VISUAL BASIC

Khái niệm

Khi tìm hiểu thấu đáo môi trường Visual Basic, bạn sẽ nhận ra rằng Visual Basic ăn ý với các chuẩn chương trình Windows. Nhiều chương trình Windows chứa menu và thanh công cụ làm việc hệt như menu và thanh công cụ của Visual Basic.

Phần trên của màn hình bao gồm thanh menu và thanh công cụ. Thanh menu gồm có danh sách menu xổ xuống giúp bạn quản lý chương trình Visual Basic. Thanh công cụ cung cấp các lệnh nhanh cho những tác vụ thông dụng.

Lưu ý

Đừng nhầm lẫn giữa thanh công cụ và hộp công cụ. Thanh công cụ hiển thị dưới thanh menu và bao gồm các nút biểu tượng. Hộp công cụ là cách gọi đặc trưng của cửa sổ Toolbox nơi các điều khiển được tìm thấy và sau đó đặt chúng trên mẫu biểu.

Thanh menu

Nếu bạn đã từng làm việc nhiều với các chương trình Windows, chắc hẳn bạn quen với lệnh trên thanh menu File, Edit, View, Window và Help bởi vì chúng giống hệt nhiều trình ứng dụng Windows khác. Bảng 2.2 mô tả tất cả các lệnh trên thanh menu Visual Basic mà bạn sẽ áp dụng.

Ghi chú

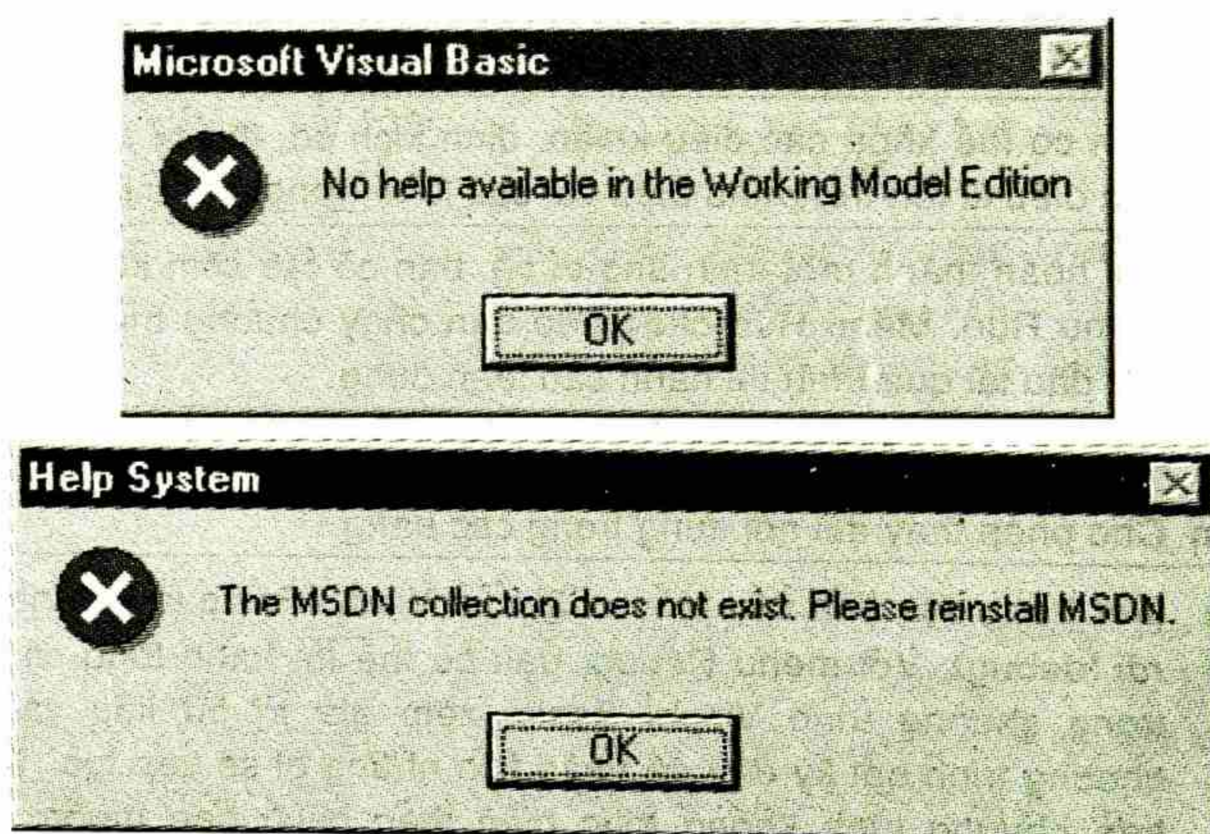
Thanh menu chứa những menu bổ sung xổ xuống giống như hầu hết các trình ứng dụng Windows đã sử dụng. Menu xổ xuống này đôi khi gọi là menu con. Lệnh trên menu con thực hiện các tác vụ hoặc đưa ra hộp thoại yêu cầu thông tin từ bạn trước khi Visual Basic có thể thi hành lệnh.

Bảng 2.2. *Các lệnh trên thanh Menu*

Lệnh	Mô tả
File	Menu File bao gồm tất cả các lệnh liên quan đến tập tin bạn có thể nạp và lưu trình ứng dụng Visual Basic. Menu còn cung cấp lệnh truy xuất in nhằm in nội dung mô tả chương trình chính xác hệt như lệnh Exit mà bạn đã tìm hiểu.

Edit	Lập trình viên thường sử dụng lệnh trên menu Edit để sao chép, cắt, dán văn bản và điều khiển đồ họa giữa các trình ứng dụng. Lệnh Edit còn giúp bạn tạo chương trình bằng cách hỗ trợ lệnh tìm kiếm và thay thế thông thường.
View	Lệnh View cho phép điều chỉnh cách nhìn cửa sổ Code trong trình ứng dụng, các thủ tục khác nhau có thể xuất hiện bên trong cửa sổ Code, cũng như thanh công cụ. Bằng cách giấu thanh công cụ, bạn có thể thêm một ít khoảng trống trên màn hình. Chẳng hạn, nếu bạn muốn vùng làm việc thoải mái hơn và không sử dụng thường xuyên thanh công cụ, có thể giấu thanh công cụ bằng cách chọn View ➥ Toolbar, xóa dấu chọn trên thanh công cụ đang hiển thị. Thanh công cụ sẽ biến mất. Chọn lại thanh công cụ bằng lệnh View ➥ Toolbar sẽ hiển thị thanh công cụ trở lại. Với menu View, bạn có thể hiển thị cửa sổ Project, Properties, và Toolbox cũng như cửa sổ Debug (nơi bạn có thể làm việc trong khi dò lỗi chương trình).
Project	Với menu Project, bạn có thể mẫu biểu, Module, điều khiển ActiveX, hay các tập tin khác tới bài thực hành.
Format	Bạn có thể khóa các điều khiển, định kích cỡ, thứ tự sắp xếp của các điều khiển trên mẫu biểu với các lệnh trong menu này.
Run	Khi hoàn thành một trình ứng dụng, bạn có thể xem kết quả bằng menu Run. Menu Run cho phép bạn chạy chương trình, dừng và bắt đầu lại quá trình thi hành sau lệnh dừng.
Query	Cho phép thiết kế và chạy các vấn tin.
Diagram	Cho phép thay đổi nội dung trong các bảng.
Debug	Một trong đặc tính tiêu biểu nhất của Visual Basic là khả năng *gỡ rối* (debug). Với menu Debug, bạn có thể thi hành từng câu lệnh trong chương trình Visual Basic, xem giá trị dữ liệu và dừng chương trình bất kỳ đâu để phân tích những gì sẽ tiếp tục. Trường hợp chương trình không thực hiện theo cách bạn muốn, menu Debug sẽ giúp bạn tìm ra nguyên nhân.
Tools	Bạn có thể xác định phương thức Visual Basic sẽ hành động bằng cách thay đổi giá trị trong menu Tools. Có thể điều khiển cả hai tùy chọn môi trường – môi trường Visual Basic trong đó bạn thiết kế các chương trình – và bổ sung tùy chọn định rõ cách thức chạy từng trình ứng dụng.

Add–Ins	Dùng để nạp các công cụ điều khiển khác như ActiveX, hỗ trợ thiết kế trình ứng dụng cao cấp trong Visual Basic.
Window	Với menu Window, bạn có thể sắp xếp lại các cửa sổ trong màn hình Visual Basic theo ý muốn.
Help	Khi chọn từ menu Help của Visual Basic, bạn sẽ không nhận được cơ chế trợ giúp trực tuyến nếu không dùng phiên bản đầy đủ, hoặc quá trình cài đặt của bạn không đúng, thiếu MSDN. Ví dụ, nếu chọn Help Contents, Visual Basic sẽ hiển thị cửa sổ như được minh họa ở Hình 2.6, mô tả hệ thống Help giống như bạn sử dụng phiên bản thông thường của Visual Basic vậy (nếu nhìn thấy màn hình này, hãy chọn OK). Một số lệnh bên dưới menu Help đưa ra hộp thông báo hầu cho biết không có giúp đỡ hiệu lực. Tuy nhiên, bạn có thể hiển thị hộp About để xem có bao nhiêu bộ nhớ có hiệu lực, cũng như thông báo bản quyền và phiên bản của hệ thống Visual Basic.



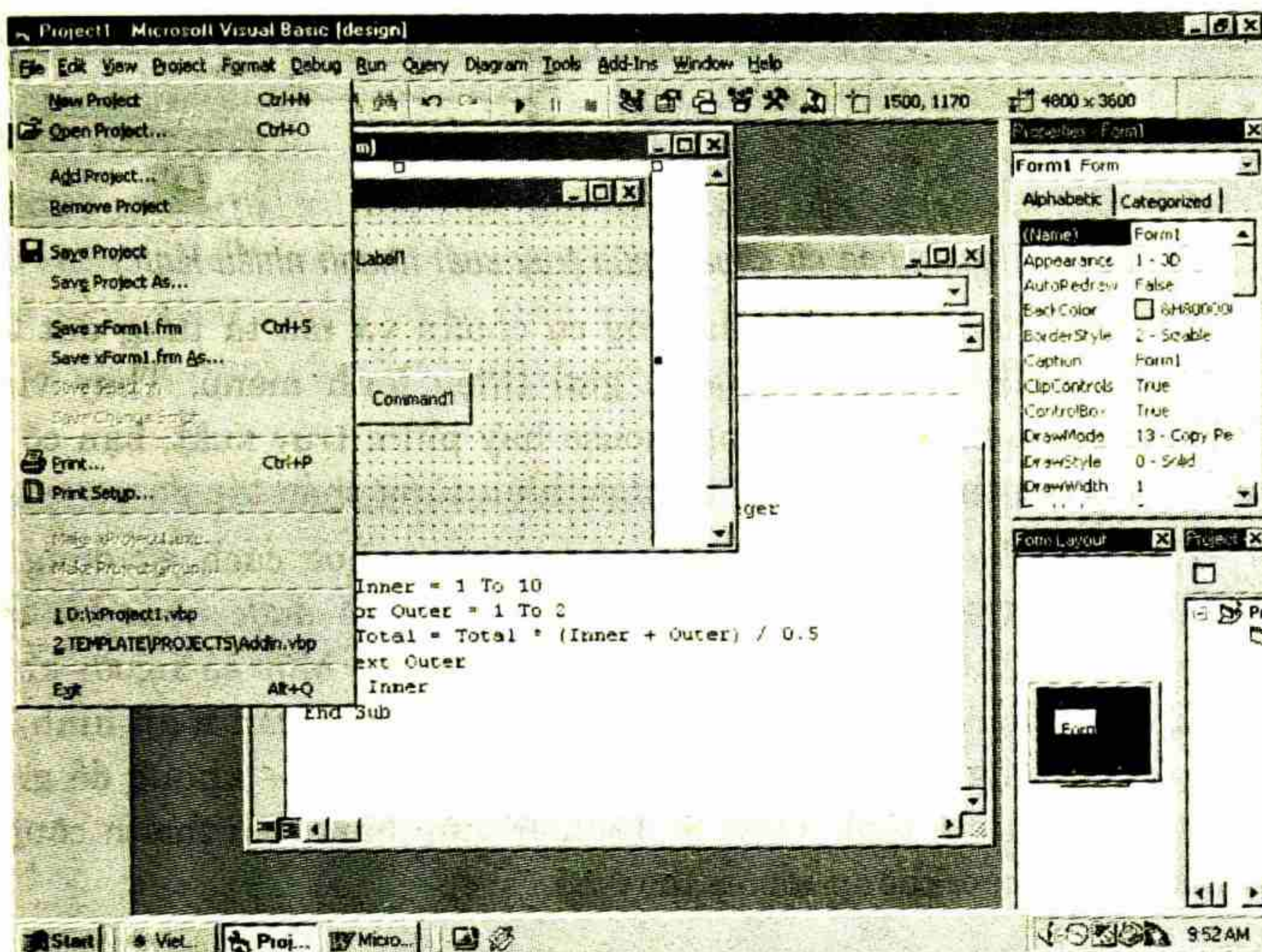
Hình 2.6. *Visual Basic sẽ không hỗ trợ cho bạn.*

Các phím truy xuất nhanh

Khái niệm mới

Tổ hợp phím truy xuất là phương pháp cho phép thực hiện nhanh các lệnh.

Nhiều lệnh trên thanh menu cũng kích hoạt khi bạn nhấn tổ hợp phím truy xuất (hay còn gọi là phím tắt, phím nóng). Chẳng hạn, nếu hiển thị menu File xổ xuống, bạn sẽ thấy menu như trong Hình 2.7. Bạn có thể kích hoạt lệnh bất kỳ trên menu File bằng cách hiển thị menu và chọn lệnh. Bạn còn có thể thực hiện lệnh bằng cách nhấn phím truy xuất.



Hình 2.7. *Các phím truy xuất làm cho quá trình chọn lệnh xác định trở nên dễ dàng hơn.*

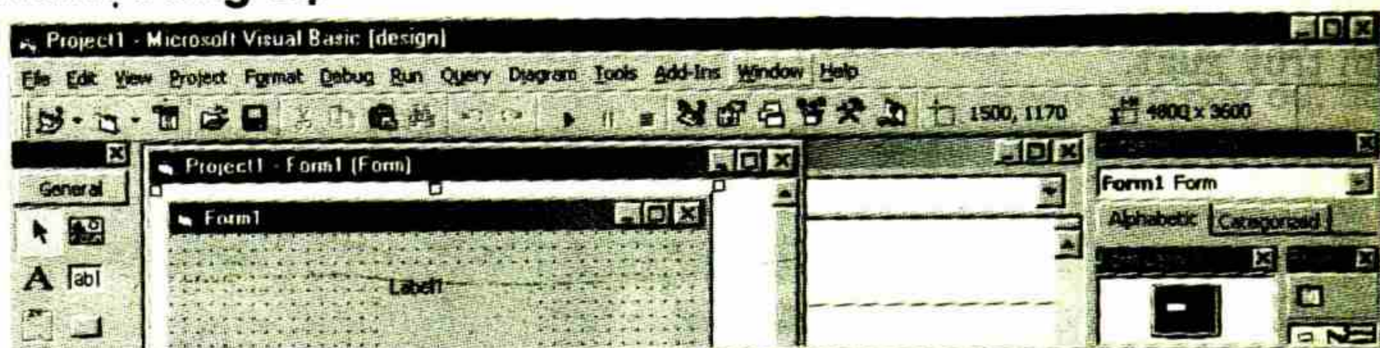
Thay vì chọn Project, Add File..., bạn có thể nhấn tổ hợp phím Ctrl+D. Thay vì chọn Save Form, bạn có thể nhấn tổ hợp phím Ctrl+S. Phím truy xuất có hiệu lực trong Visual Basic ngay cả khi trước đó bạn không hiển thị menu. Ví dụ, bạn có thể lưu tập tin đang làm việc bằng cách nhấn tổ hợp phím Ctrl+S mà không phải tốn thời gian hiển thị menu File.

Không phải tất cả các phím truy xuất đều yêu cầu bạn sử dụng phím Ctrl. Lấy ví dụ, một số lệnh trong menu View xổ xuống không nhất thiết phím thứ hai phải là Ctrl. Chẳng hạn lệnh View ➥ Object yêu cầu tổ hợp phím Shift+F7.

Mách nước

Như bạn sẽ thấy ở mục cuối, nhiều lệnh trên thanh công cụ cung cấp cùng một chức năng hệt như lệnh menu.

Thanh công cụ



Hình 2.8. *Thanh công cụ chuẩn giúp truy xuất nhanh nhiều lệnh.*

Hình 2.8 minh họa thanh công cụ chuẩn và mô tả từng nút trên nó. Nhiều nút trên thanh công cụ giới thiệu lệnh menu. Thay vì thi hành lệnh menu bằng cách dùng mouse hay phím truy xuất, bạn có thể trỏ đến nút trên thanh công cụ để thực hiện cùng thao tác đó.

Khi bạn đọc phần sau cuốn sách này và học cách sử dụng các lệnh xuất hiện trên những thanh công cụ, bạn sẽ được nhắc lại rằng khi nào mới có thể dùng nút trên thanh công cụ. Một số người không thích sử dụng thanh công cụ. Họ muốn nhiều không gian màn hình hơn và không cho rằng các biểu tượng làm cho họ dễ nhớ – không dễ gì ghi nhớ hết. Hãy nhớ là lệnh View ➥ Toolbar giúp bạn giấu thanh công cụ khỏi màn hình nếu không muốn thấy nó.

Lưu ý

Trước khi gỡ bỏ thanh công cụ chuẩn, bảo đảm rằng bạn không cần bộ chỉ dẫn đơn vị đo sẽ xuất hiện bên phải thanh công cụ chuẩn. Mục tiếp theo giải thích những gì mà bộ chỉ dẫn đơn vị đo sẽ thực hiện.

Hãy lưu ý không phải tất cả các nút trên thanh công cụ đều có màu sẫm. Một số có màu xám khi lệnh tương ứng trên thanh menu xổ xuống cũng có màu xám. Visual Basic nắm rõ các lệnh xác định có thể được kích hoạt tại thời điểm xác định trong chương trình. Ví dụ, giả như bạn không sao chép văn bản hay điều khiển vào bộ nhớ tạm, bạn không thể sử dụng lệnh Edit ➥ Paste.

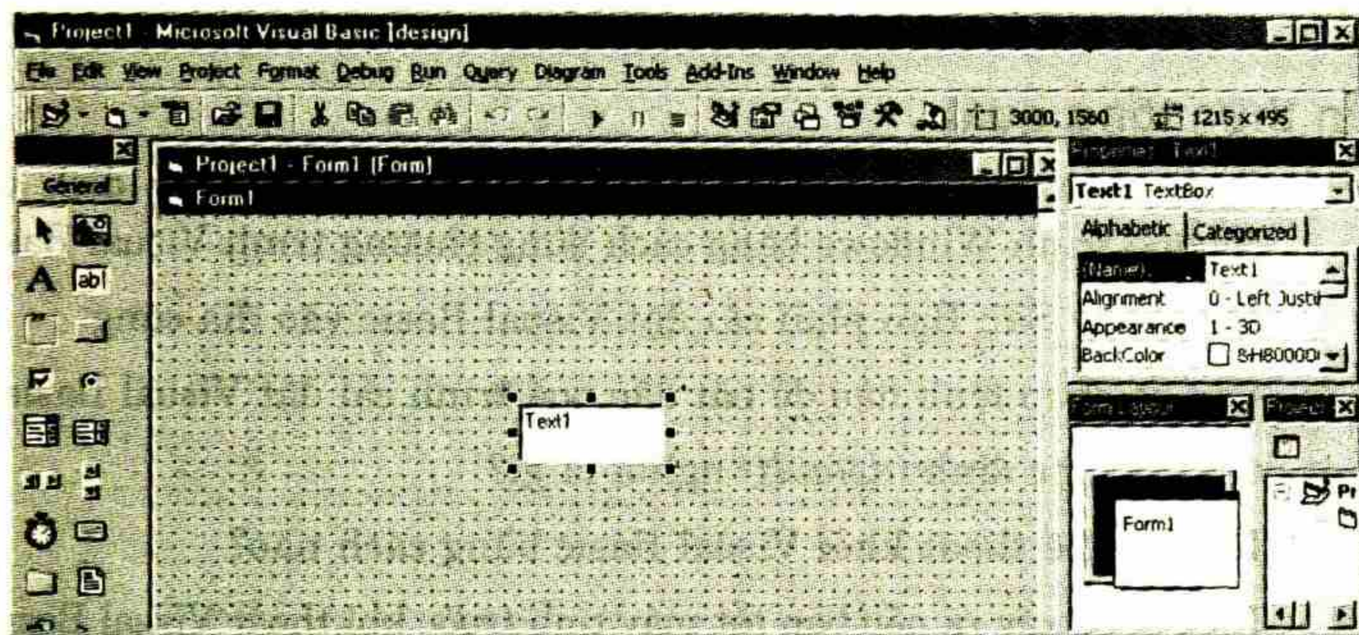
BỘ CHỈ DẪN ĐƠN VỊ ĐO

Khi vẽ và hiệu chỉnh kích cỡ ảnh từ cửa sổ Form, bạn thường thấy hai bộ chỉ dẫn đơn vị đo xuất hiện ở bên phải thanh công cụ chuẩn để trợ giúp. Công cụ đầu tiên mô tả tọa độ góc trái trên điều khiển và công cụ thứ hai mô tả kích thước điều khiển.

Khái niệm mới

1 twip *bằng 1/1440 inch.*

Đơn vị đo được tính bằng twip. Ví dụ, Hình 2.9 minh họa hộp đặt ở giữa cửa sổ Form. Bộ chỉ dẫn đơn vị đo cho biết góc trái trên hộp xuất hiện chính xác ở twip 3000 so với góc bên trái cửa sổ Form và chính xác ở twip 1560 so với phần trên cửa sổ Form. Tương tự, bạn biết rằng hộp có chiều rộng chính xác bằng 1215 twip và chiều dài là 495 twip.



Hình 2.9. *Bộ chỉ dẫn đơn vị đo cho phép bạn hiệu chỉnh kích thước và đặt điều khiển trên cửa sổ Form.*

Bằng cách sử dụng bộ chỉ dẫn đơn vị đo, bạn có thể dám chắc mọi thành phần trong trình ứng dụng được sắp xếp và hiệu chỉnh đúng kích cỡ mong muốn.

Ghi chú

Các điểm trên khung lưới cửa sổ Form trong màn hình nền giúp bạn canh chỉnh lề hình ảnh. Đôi khi khung lưới được gọi là đoạn khung lưới. Bởi lẽ điều khiển mà bạn đặt trên cửa sổ Form sẽ gắn với điểm gần nhất nếu bạn đặt điều khiển giữa hai điểm. Giả sử bạn muốn điều chỉnh khoảng cách giữa hai điểm trên khung lưới, dùng lệnh Tools ➡ Options ➡ General. Còn như bạn muốn tắt khung lưới để đặt điều khiển giữa các điểm theo ý thích, bạn có thể chuyển tùy chọn Align Controls To Grid sang No cũng bằng lệnh trên.

Củng cố

Bây giờ, bạn đã hiểu rõ toàn bộ màn hình Visual Basic và môi trường làm việc của nó. Dù cho bạn không biết cách sử dụng tất cả thành phần trong môi trường, nhưng ít nhất bạn đã làm quen với môi trường và sẽ nhận ra tên của những thành phần màn hình khi gặp lại chúng trong sách.

Bài tập

Kiến thức tổng quát

1. Đúng hay Sai: Bạn có thể chạy nhưng không biên dịch các chương trình Windows bằng cách dùng chương trình Visual Basic.
2. Đúng hay Sai: Bạn phải cài đặt Visual Basic vào đĩa cứng.
3. Bạn sẽ chọn lệnh nào để bắt đầu quá trình cài đặt Visual Basic?
4. Giá trị mặc định (*default*) nghĩa là gì?
5. Bạn có thể thoát khỏi Visual Basic bằng cách nào?
6. Điều gì sẽ xảy ra nếu bạn tắt máy tính trước khi thoát Visual Basic?
7. Tên của năm kiểu cửa sổ chính trong Visual Basic.
8. Kiểu cửa sổ nào Visual Basic sử dụng làm màn hình cho trình ứng dụng?
9. Đúng hay Sai: Không có nét khác biệt giữa hộp công cụ và thanh công cụ.
10. Đúng hay Sai: Bạn có thể giấu hộp công cụ trên màn hình.
11. Công dụng của phím truy xuất.
12. Công dụng của thanh công cụ.

13. Tại sao thanh menu của Visual Basic thông dụng đối với hầu hết người dùng Windows?
14. Bao nhiêu thao tác gõ phím truy xuất có hiệu lực trên menu Edit xổ xuống?
15. Tại sao lệnh trên menu Visual Basic và các nút trên thanh công cụ chuyển sang màu xám lúc gõ phím?
16. Bộ chỉ dẫn đơn vị đo trên thanh công cụ dùng để làm gì?
17. Bạn biết gì về *twip*?
18. Khung lưới giúp bạn chỉnh lề các điều khiển trên mẫu biểu như thế nào?

Phần nâng cao

Hãy xác định lệnh menu tương ứng với từng nút trên thanh công cụ. Điều này sẽ giúp bạn quen thuộc với menu và các lệnh của nó.

Bài thực hành 1

Chào mừng bạn đã sử dụng Visual Basic!

Ghi chú

Bài thực hành đầu tiên này có một vài khác biệt nhỏ so với các bài thực hành khác: Mặc dù đang tìm hiểu môi trường Visual Basic, nhưng bạn chưa thể hiểu được chương trình Visual Basic. Bắt đầu từ Bài Thực Hành 2, bạn sẽ thấy trọn vẹn phương thức làm việc của chương trình Visual Basic.

Tóm tắt

Suốt chương này, bạn đã biết về lịch sử Visual Basic và các nguyên tắc cơ bản trong môi trường lập trình của nó. Visual Basic là một hệ thống lập trình giàu công cụ. Năm kiểu cửa sổ trong Visual Basic vận hành kết hợp với nhau để cung cấp cho bạn các công cụ thiết kế chương trình.

Trong chương này, bạn đã tìm hiểu:

- Khái niệm về lập trình
- Khái quát ngôn ngữ BASIC
- Cách lập trình trên Windows khác biệt rất nhiều so với lập trình trên các hệ thống dựa trên văn bản như DOS chẳng hạn
- Các loại lỗi lập trình có thể gặp
- Lý do xuất hiện lỗi chương trình và các bước giải quyết
- Cách cài đặt Visual Basic
- Cách khởi động Visual Basic
- Các thành phần trên màn hình và môi trường Visual Basic, chẳng hạn năm kiểu cửa sổ cơ bản
- Các menu Visual Basic
- Cách thoát khỏi chương trình Visual Basic

Bước 1: Khởi động Visual Basic

Trước khi viết một chương trình Visual Basic, bạn phải khởi động hệ thống Visual Basic. Hãy thực hiện các bước sau:

1. Bật máy tính.
2. Khởi động Windows.
3. Mở nhóm chương trình Microsoft Visual Basic 6.0 và nhấp đúp biểu tượng Microsoft Visual Basic 6.0. Môi trường lập trình Visual Basic sẽ xuất hiện.

Bước 2: Thực hành với Visual Basic

1. Định lại kích cỡ cửa sổ Form để góc dưới của nó tiếp cận cuối màn hình. Di chuyển con trỏ mouse tới góc dưới cửa sổ Form cho đến khi con trỏ mouse trở thành một mũi tên hai đầu. Kéo góc dưới xuống một đoạn sao cho bạn có thể kéo và nhả nút mouse.
2. Khi nhấp góc dưới cửa sổ Form, cửa sổ Form có lẽ sẽ che các cửa sổ Project và Properties khỏi tầm nhìn. Nhấn tổ hợp phím Alt+V, P để mang cửa sổ Project trở lại tầm nhìn. Bằng cách nhấn tổ hợp phím Alt+V, P, bạn đã chọn lệnh menu View ➥ Project Explorer .
3. Cửa sổ Toolbox là cửa sổ bạn sẽ sử dụng thường xuyên với cửa sổ Form. Tuy nhiên, có lẽ bạn không thích vị trí của cửa sổ Toolbox. Thỉnh thoảng, khi trải dài cửa sổ Form lên toàn màn hình, bạn muốn đặt các mục ở bên trái cửa sổ Form, nơi Toolbox đang hiển thị. Vì vậy, hãy thực hành việc di chuyển cửa sổ Toolbox đi nơi khác.

Đưa con trỏ mouse tới thanh tiêu đề ở đầu cửa sổ Toolbox. Thanh này nằm bên phải nút điều khiển nên gọi là thanh tiêu đề, ngay cả khi không có tiêu đề nào trong cửa sổ Toolbox. Nhấn–giữ mouse, xong kéo cửa sổ Toolbox tới bên phải màn hình.

Bước 3: Sửa đổi môi trường

1. Nếu bạn muốn giải phóng các thanh công cụ – là thanh chứa những nút biểu tượng ngay dưới thanh menu, hãy nhấn tổ hợp phím Alt+V, và nhấn Enter trên thanh công cụ muốn che giấu. Thanh công cụ sẽ biến khỏi màn hình.

2. Visual Basic cho phép bạn tắt các khung lưới trên cửa sổ Form. Nhấn tổ hợp phím Alt+T, O để chọn lệnh Tools ➡ Options, sau đó chọn mục General, nhấp ô chọn Show Grid để bỏ chọn. Hoặc nhấn Tab chuyển đến ô chọn, rồi nhấn phím Spacebar.
3. Nhấn Enter hay nhấp OK để xem kết quả. Cửa sổ Form không còn khung lưới nữa.
4. Bạn có cảm thấy lạc quan với những kỹ xảo Visual Basic của bạn không? Hãy đảo lộn mọi thay đổi mà bạn đã thực hiện trong phần này. Bật trở lại thanh công cụ chuẩn và các khung lưới.

Ghi chú

Nếu bạn muốn làm việc nhiều hơn trong môi trường Visual Basic, cứ việc tiếp tục. Khi đã hoàn thành, thoát khỏi Visual Basic bằng cách nhấn tổ hợp phím Alt+F, X. Visual Basic sẽ báo rằng bạn đã thực hiện các thay đổi và sẽ hiển thị một hộp thoại hỏi bạn có muốn lưu những thay đổi cho mẫu biểu không. Chọn No để báo cho biết bạn không muốn giữ lại bất kỳ thay đổi nào trong bài thực hành này và muốn trở lại Windows. Bạn luôn luôn trở lại Windows sau khi thoát khỏi Visual Basic.

Chương II

Bài 3

Nội dung chương trình Visual Basic

- **Nạp và chạy chương trình**
- **Điều khiển nhãn**
- **Điều khiển hộp nhập**
- **Các nút lệnh thật thú vị !**
- **Điều khiển ô chọn**
- **Nút tùy chọn giới hạn sự chọn lựa**
- **Tạo khung**
- **Danh sách combo xổ xuống**
- **Hộp combo đơn giản**
- **Hộp danh sách với quá trình lựa chọn**

Bài này giải thích tất cả dạng mẫu về chương trình Visual Basic. Không giống phần lớn các ngôn ngữ lập trình, ngôn ngữ Visual Basic gồm có mã lệnh và các điều khiển đồ họa. Mã lệnh chương trình tương tác với điều khiển để xuất ra kết quả và tương tác với người dùng.

Bài này tập trung về các điều khiển đồ họa có thể sử dụng trong chương trình Visual Basic. Khi đọc những chương sau, bạn sẽ nắm được cách làm việc với mã lệnh. Một số lệnh Visual Basic tương đối dễ, nhưng một số khác lại phức tạp. Bài này tập trung vào giải thích nội dung của chương trình. Nó chỉ cho bạn cách nạp và chạy các chương trình Visual Basic.

Ghi chú

Đây là một bài thực hành. Bạn cần sử dụng chương trình Visual Basic khi đọc bài này để học cách làm việc của từng công cụ đồ họa trong Visual Basic.

NẠP VÀ CHẠY CHƯƠNG TRÌNH

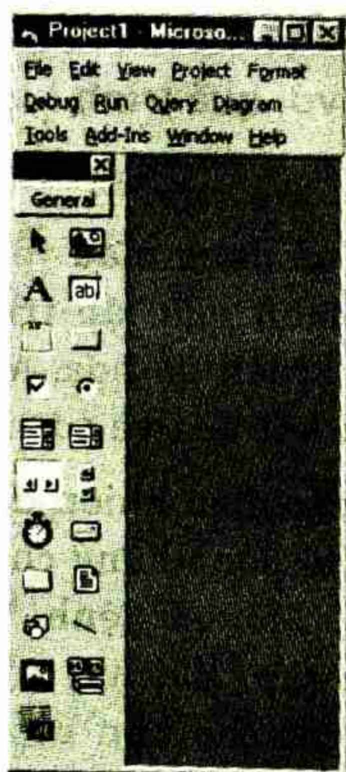
Khái niệm

Một khi khởi động Visual Basic, bạn có thể tạo một chương trình. Sau khi tạo chương trình, bạn dễ dàng cất chương trình đó, rồi nạp và chạy nó sau. Bài này chỉ cho bạn cách nạp và chạy chương trình được lưu trên đĩa.

Khái niệm mới

Người dùng thực hiện các thao tác nhập xuất dữ liệu thông qua các điều khiển.

Bài này sẽ trình bày nhiều điều khiển quan trọng trong Visual Basic. Điều khiển quản lý nhiều thao tác nhập xuất của người dùng, thường gọi là *I/O*. Nói cách khác, khi người dùng chạy chương trình bạn tạo, họ phải tương tác với chương trình bằng cách trả lời các câu hỏi và lựa chọn những tùy chọn hiển thị trên màn hình.



Hình 3.1. *Các điều khiển mà chương trình của bạn cần sử dụng trên cửa sổ Toolbox.*

Hãy nhớ rằng cửa sổ Toolbox là nơi bạn lấy các điều khiển để đặt lên cửa sổ Form. Cửa sổ Form là màn hình nền của ứng dụng bạn tạo. Hình 3.1 mô tả các điều khiển trên cửa sổ Toolbox.

Tập tin project: Một chương trình Visual Basic hiếm khi chỉ lưu trữ trong một tập tin. Cần có nhiều tập tin để mô tả một trình ứng dụng Visual Basic. Tập tin project giúp chia nhỏ từng phần trong chương trình Window thành nhiều tập tin.

Ví dụ, đa số trình ứng dụng có tập tin project. Tập tin mẫu biểu lưu nội dung cửa sổ Form mà bạn tạo cũng như mã lệnh xử lý các đối tượng trên cửa sổ Form. Có thể bạn cũng viết mã lệnh bổ sung và lưu nó trong một tập tin riêng biệt. Nếu trình ứng dụng của bạn yêu cầu các điều khiển đặc biệt không xuất hiện thường xuyên trên cửa sổ Toolbox, bạn phải thêm các tập tin điều khiển này vào trình ứng dụng.

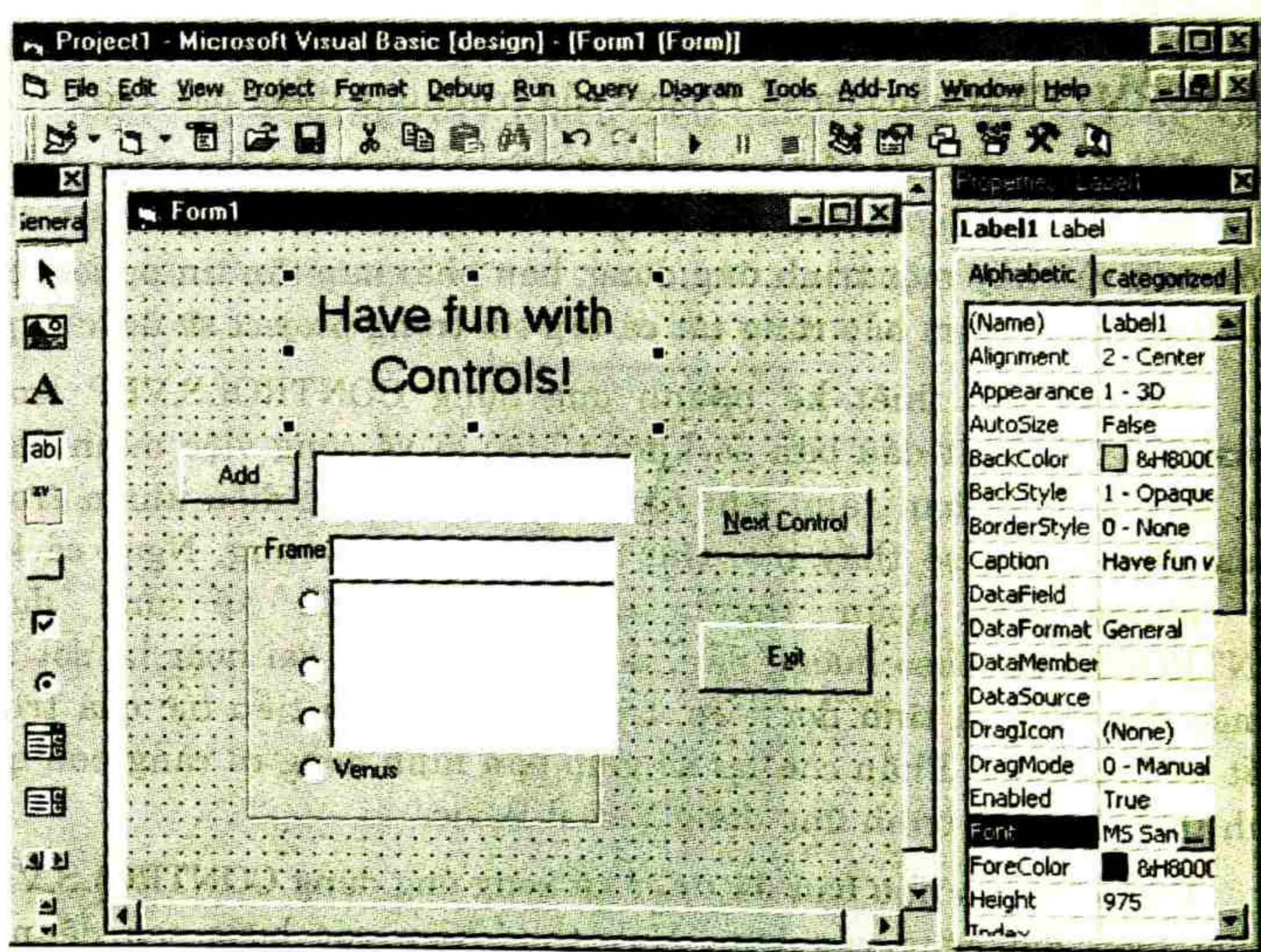
Cửa sổ Project gồm có danh sách các tập tin trong trình ứng dụng. Mặc định, Visual Basic luôn thêm tập tin project với tên mặc định FORM1.FRM vào cửa sổ Project của trình ứng dụng mới tạo.

Nội dung của cửa sổ Project được mô tả trong tập tin project. Mỗi trình ứng dụng đều có tập tin project với phần mở rộng .VBP. Khi muốn nạp và chạy một trình ứng dụng, bạn phải nạp tập tin project của nó; Visual Basic đảm bảo rằng tất cả tập tin trong project sẽ được nạp.

Bạn có thể thiết kế nhanh ứng dụng CONTROLS.VBP trong Visual. Có lẽ, thoạt đầu bạn sẽ không hiểu gì về công việc mình đang làm, nhưng đừng bận tâm. Khi bạn đã làm chủ các điều khiển trong Visual Basic, bạn sẽ thấy mọi thứ trở nên thật rõ ràng. Ngay cả khi bạn không thể hoàn thành quá trình thiết kế ứng dụng CONTROLS.VBP theo hướng dẫn, do bạn thực hiện sai hoặc bỏ sót tác vụ nào đó, cứ bình tĩnh đọc tiếp, trước khi quay lại tiếp tục quá trình thiết kế ứng dụng. Phần còn lại sẽ giúp bạn hình dung rõ ràng hơn quá trình thiết kế một trình ứng dụng Visual Basic.

Bây giờ, xem như bạn đã thiết kế xong ứng dụng CONTROLS.VBP và lưu trên đĩa cứng của bạn. Hãy thực hiện các bước sau để nạp CONTROLS.VBP:

1. Khởi động Visual Basic.
2. Chọn File ➥ Open Project từ thanh menu. Visual Basic hiển thị cửa sổ hộp thoại File Open tiêu chuẩn.
3. Chọn tên ổ đĩa và đường dẫn thư mục lưu tập tin project CONTROLS.VBP. Chọn hay nhập tên tập tin project sau ở dấu nhắc File Name: **CONTROLS.VBP**. Bạn có thể nhập tên ở dạng chữ thường hoặc chữ hoa.
4. Visual Basic sẽ nạp project. Khi Visual Basic tìm thấy tập tin project CONTROLS.VBP, nó đọc tập tin và nạp tất cả tập tin liên quan tới project đó. Bạn có thể thấy từ cửa sổ Project, Visual Basic đã nạp tập tin project.
5. Để xem mẫu biểu, nhấn nút View Form trên cửa sổ Project. Màn hình của bạn trông giống Hình 3.2.

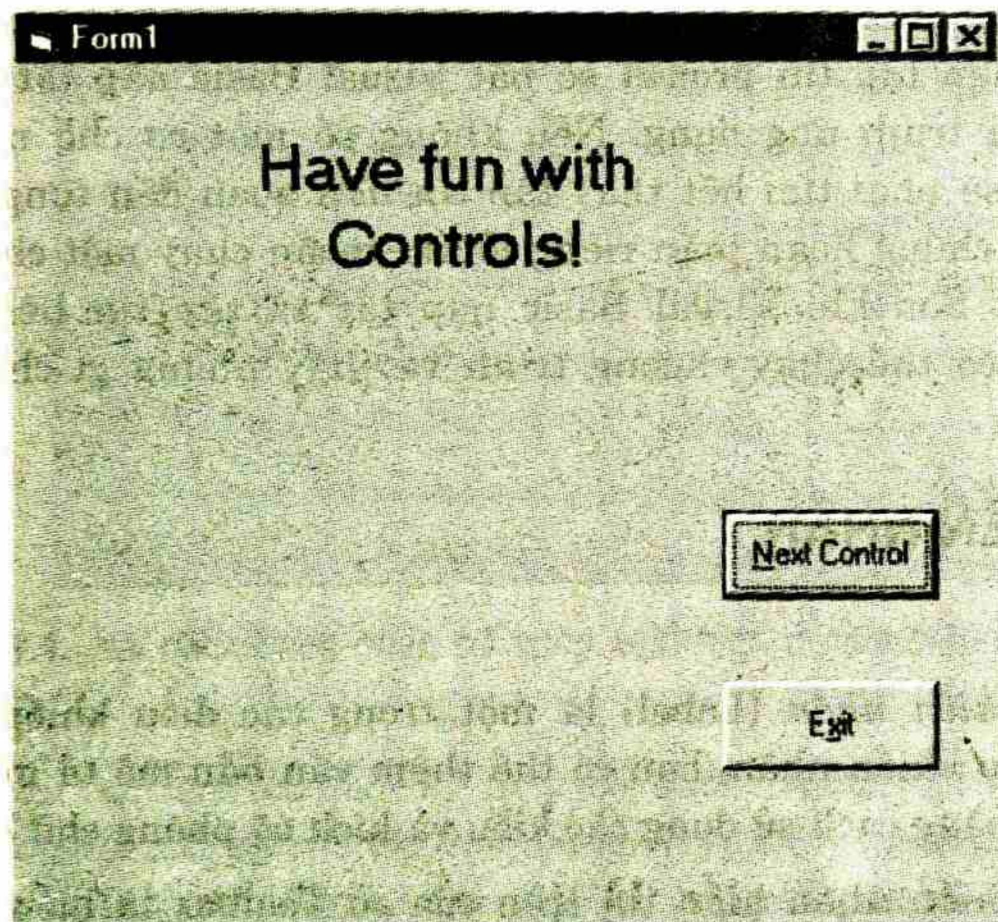


Hình 3.2. *Cửa sổ Form trông rất chật.*

Khi người dùng chạy chương trình, họ sẽ thấy cửa sổ Form. Tuy nhiên, những gì người dùng thấy không giống cửa sổ Form khi chương trình được nạp lần đầu tiên. Nói cách khác, bây giờ bạn đang xem mọi thành phần trong chương trình chất đầy trên một cửa sổ. Trong trường hợp này, khi người dùng chạy chương trình, chương trình bảo đảm rằng các thành phần này sẽ hiển thị theo trình tự logic và không chồng lấp lên nhau.

Trước khi chạy chương trình trong môi trường Visual Basic, bạn phải nạp chương trình với lệnh File ➥ Open. Một khi đã nạp chương trình, bạn sẽ có ba cách để chạy nó:

- Chọn lệnh Run Start từ thanh menu.
- Nhấn phím F5, phím truy cập nhanh của lệnh Run Start.
- Nhấn nút Start trên thanh công cụ. (Như bạn đã thấy trong bài học trước, nút Start trông giống nút Play trên máy cassette.)



Hình 3.3. *Chương trình đang chạy rõ ràng hơn nhiều.*

Dùng một trong ba phương pháp này để chạy chương trình. Bạn sẽ thấy màn hình có trật tự và ít lộn xộn hơn Hình 3.2. Các lệnh khởi tạo chương trình làm sạch mớ hỗn độn bạn đã thấy lúc đầu trong cửa sổ Form. Giờ đây, cửa sổ Form của môi trường Visual Basic trở thành màn hình nền của trình ứng dụng (ngoại trừ các khung lưới trên nền).

Chương trình này minh họa các điều khiển Visual Basic chính để bạn làm quen với cách sử dụng chúng trước khi đưa chúng vào trình ứng dụng riêng của mình.

Mách nước

Hãy chú ý rằng chương trình bao gồm nút lệnh Exit ở góc phải dưới. Tất cả trình ứng dụng trong sách đều có nút lệnh Exit để bạn có thể kết thúc chương trình bất cứ khi nào. Lúc bạn tạo các chương trình riêng cho mình, hãy luôn cho người dùng một cách thoát chương trình của bạn.

Củng cố

Trước khi chạy một chương trình trong Visual Basic, bạn phải nạp nó. Thông thường, bạn nạp tập tin project, có phần mở rộng là .VBP. Việc nạp tập tin project sẽ bắt Visual Basic nạp tất cả tập tin liên quan tới trình ứng dụng. Nếu không có một cơ chế như tập tin project, bạn sẽ phải tìm hết thảy tập tin liên quan đến từng trình ứng dụng và nạp chúng hoàn toàn riêng lẻ khi muốn chạy một chương trình Visual Basic. Sau khi Visual Basic nạp tất cả tập tin liên quan tới project, bạn có thể chạy chương trình và thấy những gì chương trình thực hiện.

ĐIỀU KHIỂN NHÃN

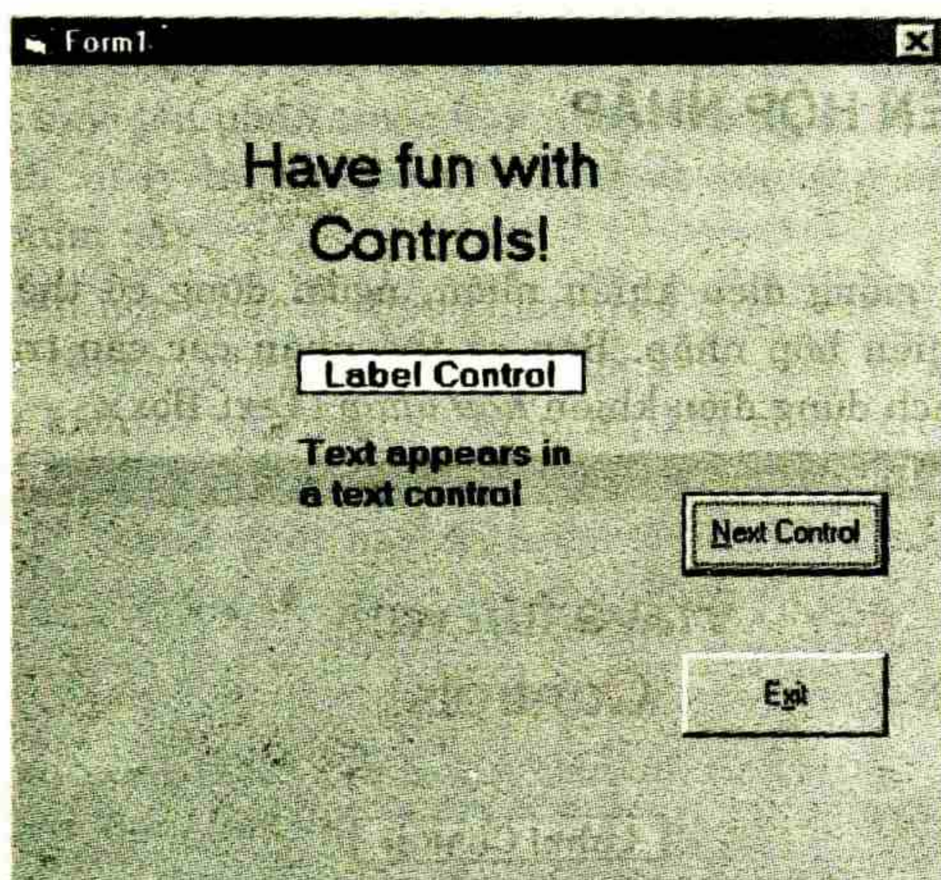
Khái niệm

Điều khiển nhãn (Label) là một trong các điều khiển đơn giản nhất. Với điều khiển nhãn, bạn có thể thêm văn bản mô tả mẫu biểu tại vị trí bất kỳ bằng cách sử dụng các kiểu và kích cỡ phông chữ khác nhau.

Điều khiển nhãn hiển thị trên cửa sổ Toolbox là biểu tượng chữ A hoa. Người dùng sẽ thấy văn bản trong kết quả trình ứng dụng. Tiêu đề của ứng dụng CONTROLS.VBP là **Have fun with controls!** được đặt trên mẫu biểu với điều khiển nhãn.

Rồi bạn sẽ thấy, bạn có thể định nhiều giá trị thuộc tính khi đặt điều khiển. Bạn sẽ quản lý kích cỡ điều khiển và cách thấy được điều khiển. Có thể xác định kích thước phông chữ lớn hay nhỏ, chọn kiểu chữ thuộc bất kỳ kiểu phông chữ nào đang có hiệu lực trong hệ thống Windows, định kiểu chữ đậm, nghiêng hay gạch bỏ với một đường thẳng ngang qua văn bản.

Bạn có thể vẽ một hộp quanh nhãn, tạo bóng và tô màu nền cho nó nếu muốn. Hãy nhìn lại chương trình đang chạy. Bạn có thể thấy nút lệnh có nhãn là Next Control. Nhấp nút lệnh này bằng mouse hay nhấn tổ hợp phím Alt+C. Hai nhãn mới sẽ xuất hiện ở giữa màn hình như Hình 3.4.



Hình 3.4. *Hai điều khiển nhãn bổ sung xuất hiện ở giữa biểu mẫu.*

Mách nước

Giả như nút lệnh có một ký tự gạch dưới, chẳng hạn nút lệnh Next Control ở góc phải cửa sổ chương trình trong Hình 3.4 – ký tự gạch dưới trình bày phím truy xuất nhanh. Bạn có thể nhấn kết hợp phím Alt với phím truy xuất nhanh để kích hoạt nút lệnh.

Lưu ý

Người dùng không thể thay đổi trực tiếp nội dung văn bản trên nhãn. Thông qua mã lệnh, bạn có thể thay đổi nội dung của nhãn để đáp ứng hành động của người dùng khi cần. Tuy nhiên, bạn thường đặt văn bản vào trong nhãn lúc thiết kế chương trình.

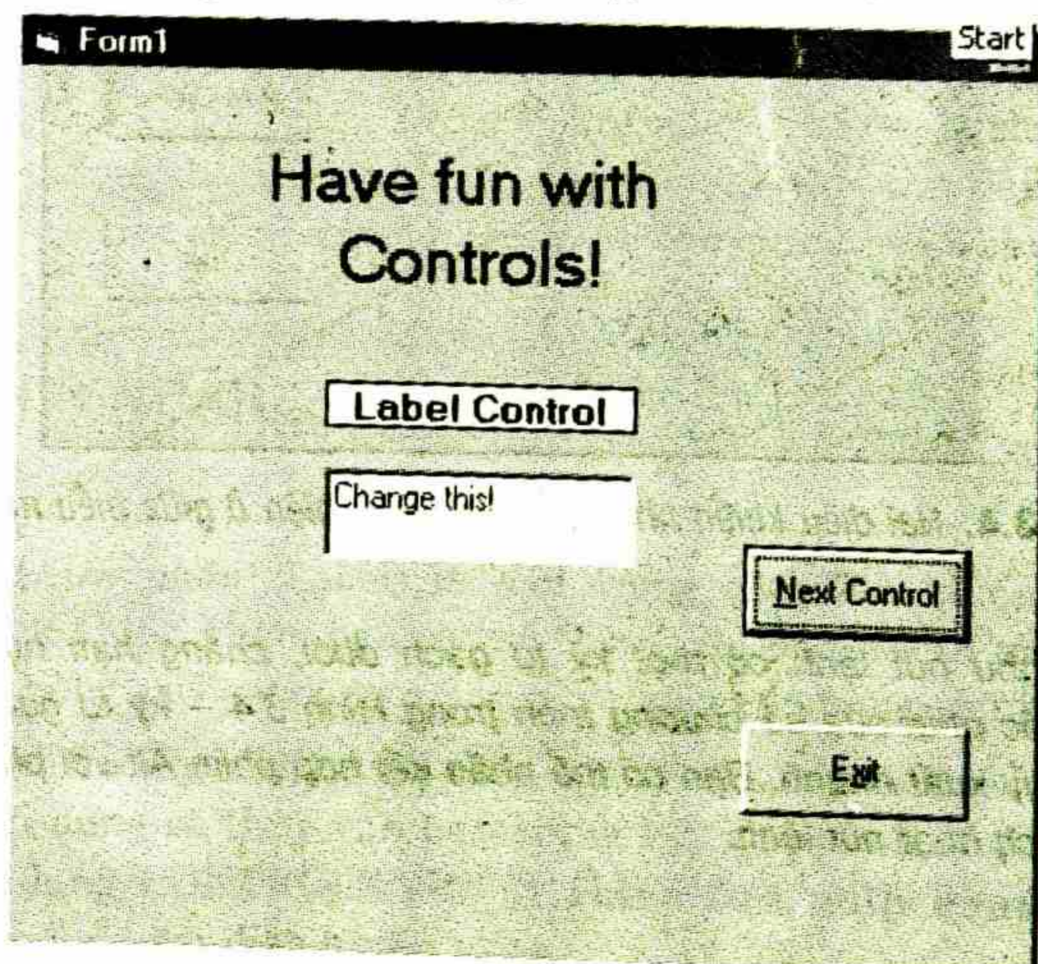
Củng cố

Điều khiển nhãn (Label) là một trong các điều khiển dễ bổ sung nhất vào các trình ứng dụng. Bất cứ khi nào bạn cần hiển thị văn bản trong một tiêu đề hay mô tả về điều khiển, bạn có thể sử dụng điều khiển nhãn. Nó cho phép bạn hiển thị văn bản với kích cỡ và kiểu phông chữ khác nhau.

ĐIỀU KHIỂN HỘP NHẬP

Khái niệm

Không giống điều khiển nhãn, người dùng có thể thay đổi nội dung điều khiển hộp nhập. Bạn có thể nhận các câu trả lời từ người dùng bằng cách dùng điều khiển *hộp nhập* (Text Box).



Hình 3.5. *Người dùng có thể thay đổi văn bản trong một hộp nhập.*

Khi nhấp nút lệnh Next Control vào thời điểm chương trình đang chạy, các nhãn đang hiển thị trước đó biến mất và hộp nhập sẽ thay thế, như Hình 3.5.

Lời khuyên

Nếu bạn hiển thị một hộp nhập trống, người dùng có thể nhập văn bản trong điều khiển *hộp nhập* (Text Box) để trả lời câu hỏi của bạn. Tuy nhiên, thỉnh thoảng bạn sẽ hiển thị văn bản ban đầu trong hộp nhập như một giá trị mặc định cho người dùng, việc thay đổi văn bản chỉ thực hiện nếu giá trị mặc định không phù hợp yêu cầu.

Nhấp con trỏ mouse vào vị trí bất kỳ trong hộp nhập Change this!. Nhập vặn bản mới vào hộp. Nếu bạn nhập văn bản dài, văn bản sẽ cuộn sang phải. Bạn có thể sử dụng các phím mũi tên để di chuyển con trỏ trở lại hay đến phần trước hộp nhập. Tương tự các phím Ins và Del trong trình xử lý từ, bạn có thể chèn và xóa văn bản trong hộp nhập với hai phím này.

Điều khiển hộp nhập là điều khiển trên hộp công cụ có ký hiệu ở trong một hộp. Nó cho phép bạn đặt nội dung ban đầu và xác định kiểu chữ, cũng như kích cỡ văn bản sẽ hiển thị khi chạy, trước khi người dùng điều chỉnh nội dung hộp nhập. Bạn có thể định cho điều khiển hộp nhập có các thanh cuốn ngang và dọc để người dùng có thể cuộn văn bản dài thay vì dùng các phím mũi tên thông thường.

Củng cố

Khi người dùng phải nhập văn bản mới hay thay đổi văn bản đang tồn tại, hãy dùng điều khiển hộp nhập để ấn định một vùng trên màn hình nhận các thao tác gõ phím của người dùng. Điều khiển hộp nhập (Text Box) làm việc giống hệt trình xử lý từ.

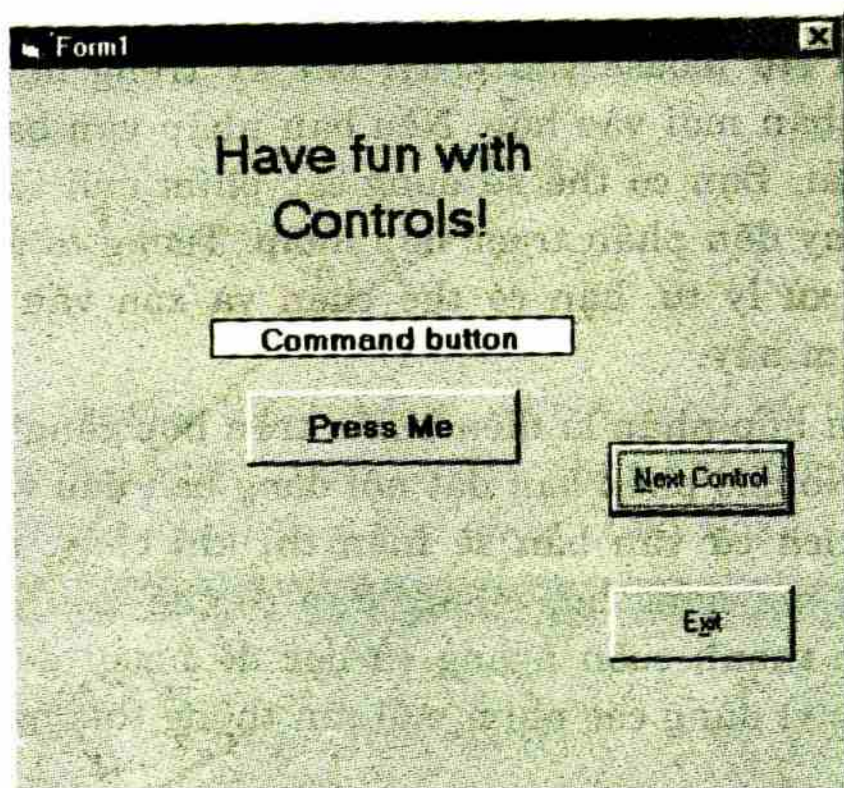
CÁC NÚT LỆNH THẬT THÚ VỊ!

Khái niệm

Bạn đã thấy các nút lệnh trong nhiều chương trình Windows, kể cả Visual Basic và ứng dụng CONTROLS.VBP đang chạy. Các nút lệnh đợi thao tác nhấn phím của người dùng để truy xuất tới các biến cố đặt trong chương trình.

Chương trình CONTROLS.VBP có hai nút lệnh sẽ tồn tại trên màn hình suốt thời gian chạy ứng dụng: đó là Next Control và Exit. Bạn có thể kích hoạt một nút lệnh bằng cách nhấp mouse hay dùng một tổ hợp phím truy xuất nhanh.

Bây giờ nhấp nút lệnh Next Control để xem nút lệnh thứ ba thay thế điều khiển hộp nhập ở giữa màn hình. Hình 3.6 mô tả màn hình sau khi nút lệnh thứ ba hiển thị.



Hình 3.6. *Nút lệnh được điều khiển bởi chương trình.*

Hầu như bất kỳ điều gì cũng có thể xảy ra khi người dùng nhấp một nút lệnh. Lập trình viên là người xác định chính xác những gì sẽ xảy ra. Tiếp tục nhấn nút lệnh trong ứng dụng CONTROLS.VBP. Hãy nhìn và lắng nghe những gì xảy ra. Máy tính phát tiếng bíp và nhãn nút lệnh thay đổi từ **Press Me** thành **Once Again**. Nhấn nút lệnh một lần nữa, lại nghe tiếng bíp và nút lệnh phục hồi về tình trạng **Press Me** ban đầu.

Bạn thường định nhãn cho nút lệnh lúc thiết kế hoặc viết chương trình. Có thể viết chương trình để thay đổi nhãn nút lệnh khi người dùng nhấp nút.

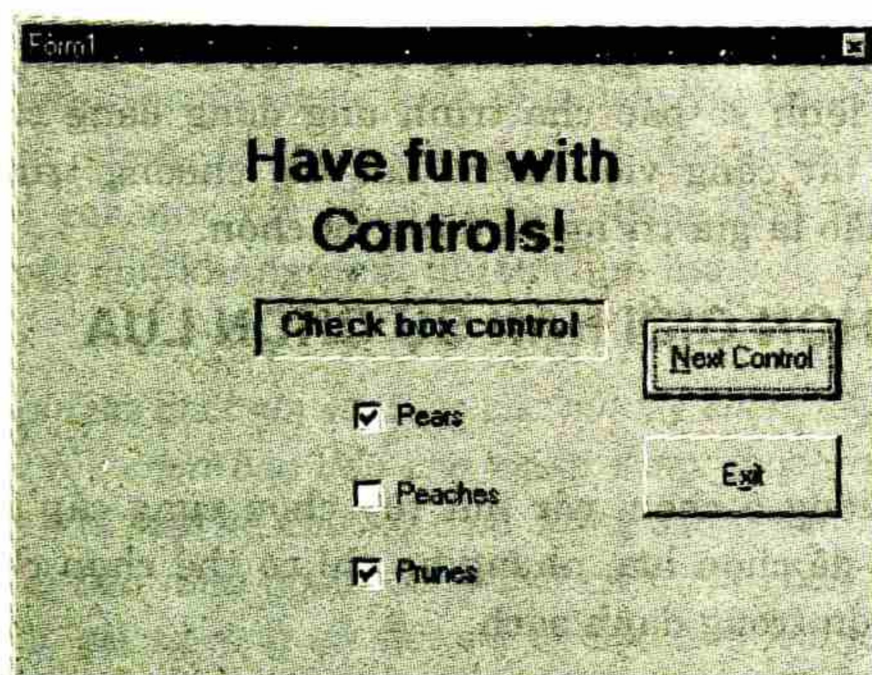
Củng cố

Một trình ứng dụng Windows không có các nút lệnh chẳng khác nào ngày không có đêm. (Vâng, dù điều đó hơi cường điệu.) Các nút lệnh đợi thao tác nhấn nút để truy xuất tới biến cố chương trình muốn người dùng kích hoạt.

ĐIỀU KHIỂN Ô CHỌN

Khái niệm

Điều khiển ô chọn (Check Box) cung cấp nhiều giá trị để người dùng chọn. Sau khi người dùng chọn một hay nhiều ô chọn, chương trình của bạn có thể phân tích ô được chọn và tiến hành xử lý dựa trên kết quả nhận được.



Hình 3.7. *Hai trong ba ô chọn được chọn.*

Nhấp nút lệnh Next Control để xem danh sách ba giá trị ô chọn trong Hình 3.7. Ô chọn chỉ cho người dùng cách chọn một hay nhiều giá trị từ danh sách các giá trị đang hiển thị.

Nhấp mouse trên ô chọn đầu tiên và cuối cùng trong màn hình nền của ứng dụng CONTROLS.VBP đang chạy (xem Hình 3.7). Khi bạn chọn ô chọn, các hộp chọn ở bên trái được đánh dấu √.

Sau khi nhấp hai điều khiển này, hai giá trị đã đề cập được chọn, giá trị ở giữa không được chọn. Bạn có thể bỏ chọn một ô chọn bằng cách nhấp lại nó lần thứ hai. Nhấp ô chọn Pears để bỏ chọn. Dấu √ sẽ biến khỏi ô chọn đó.

Mách nước

Có thể nhấp không chỉ trên ô chọn mà còn ở bất kỳ nơi đâu trong nội dung mô tả văn bản của nó. Vì vậy, để chọn hay bỏ chọn ô chọn Pears, bạn có thể nhấp con trỏ mouse trên hộp chọn trước Pears hay nhấp trên từ Pears.

Cũng có cách chọn ô chọn mà không dùng mouse. Người dùng có thể nhấn phím Tab cho đến khi mô tả ô chọn hiện sáng. Nhấn phím Spacebar để chọn nó.

Củng cố

Các ô chọn cung cấp cho người dùng nhiều lựa chọn để chọn những gì họ cần. Người dùng chọn hay bỏ chọn các ô chọn bằng mouse hoặc bàn phím. Sau khi người dùng chọn tất cả ô chọn cần thiết, việc nhấn một nút lệnh sẽ báo cho trình ứng dụng đang chạy rằng người dùng đã hoàn tất công việc chọn. Lúc đó, chương trình sẽ kiểm tra những giá trị nào là giá trị người dùng đã chọn.

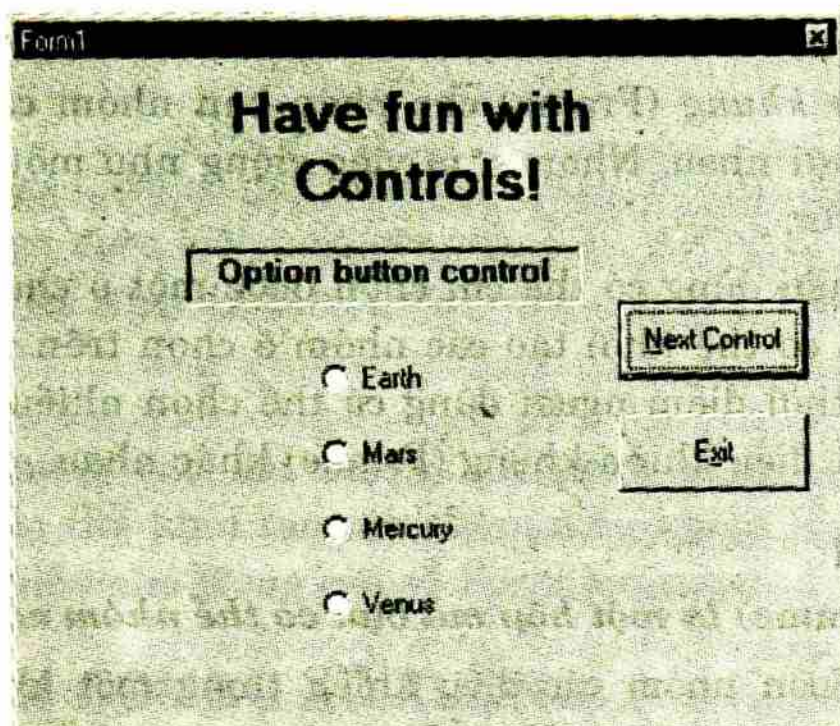
NÚT TÙY CHỌN GIỚI HẠN SỰ CHỌN LỰA

Khái niệm

Không giống ô chọn, các nút tùy chọn cung cấp cho người dùng một danh sách để chọn lựa, nhưng họ chỉ có thể chọn chính xác không quá một tùy chọn trong danh sách.

Thông thường, *điều khiển nút tùy chọn* (Option Button) được biết như điều khiển loại trừ lẫn nhau. Trừ khi bạn nhóm các tùy chọn trong các khung (sẽ được trình bày sau), tại một thời điểm người dùng chỉ có thể chọn một và chỉ một tùy chọn mà thôi.

Nhấp nút lệnh Next Control để xem các tùy chọn trên màn hình ứng dụng CONTROLS.VBP như mô tả trong Hình 3.8. Ban đầu, không có tùy chọn nào được chọn.



Hình 3.8. *Nút tùy chọn làm việc giống hệt ô chọn theo kiểu loại trừ lẫn nhau.*

Nhấp mouse trên một trong các tùy chọn. Sau đó nhấp tùy chọn khác. Ngay lập tức, Visual Basic bỏ nút đã chọn đầu tiên và chọn nút vừa nhấp. Hãy chọn nút tùy chọn khác để thay đổi chọn lựa một lần nữa. Như bạn thấy, Visual Basic luôn đảm bảo với bạn rằng chỉ có thể chọn một nút tùy chọn. Để thay cho ô chọn cho phép nhiều chọn lựa, bạn sẽ hiển thị các nút tùy chọn trong trường hợp bạn muốn người dùng chỉ chọn một chứ không phải nhiều chọn lựa.

Mách nước

Hãy suy nghĩ đôi chút khi sử dụng các nút tùy chọn. Chẳng hạn, bạn có thể chọn một nút tùy chọn ban đầu cho người dùng, thông qua mã lệnh hay cửa sổ Properties, khi thiết kế chương trình. Bằng cách chọn một nút tùy chọn ban đầu, bạn đảm bảo rằng người dùng biết cách lựa chọn tốt nhất ứng với trường hợp đã cho, giả sử rằng có một giá trị mặc định khá tốt so với các giá trị còn lại mà người dùng có thể chọn.

Củng cố

Nút tùy chọn làm việc tương tự ô chọn ngoại trừ việc người dùng có thể chọn tối đa một nút tùy chọn, thay vì ô chọn cho phép nhiều chọn lựa.

TẠO KHUNG

Khái niệm

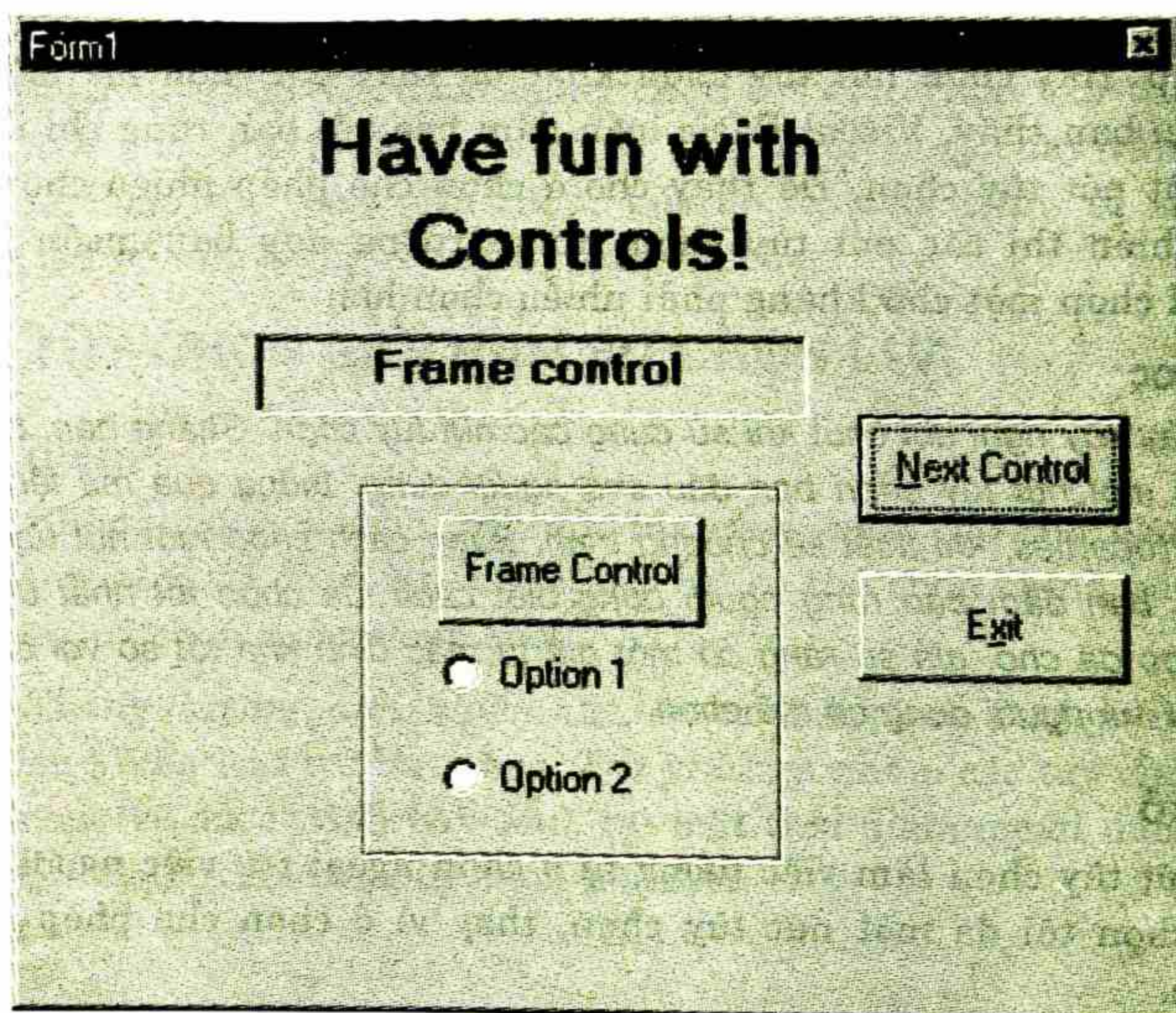
Điều khiển khung (Frame) cho phép bạn nhóm các thành phần trong mẫu biểu với nhau. Nhóm làm việc giống như một mẫu biểu nhỏ trong mẫu biểu lớn.

Mặc dù người dùng có thể chỉ chọn được một ô chọn tại một thời điểm, nhưng bạn có thể khởi tạo các nhóm ô chọn trên cùng mẫu biểu. Lúc đó, tại một thời điểm người dùng có thể chọn nhiều ô chọn, mỗi ô chọn ứng với một điều khiển khung (Frame) khác nhau.

Khái niệm mới

Khung* (Frame) *là một hộp mà bạn có thể nhóm các điều khiển.

Bạn phải luôn nhóm các điều khiển trong một khung bằng điều khiển khung. Nếu nhấp lại nút Next Control, bạn sẽ thấy một khung hiển thị ở giữa mẫu biểu, như minh họa trong Hình 3.9. Bạn có thể đặt bất kỳ điều khiển nào trong một khung, không chỉ với các ô chọn như trong hình.



Hình 3.9. *Khung gồm một nút lệnh và hai nút tùy chọn.*

Nếu có ba khung nhóm các ô chọn khác nhau hiển thị trên mẫu biểu, bạn có thể có tối đa ba lựa chọn trên mẫu biểu. Một lựa chọn trong một khung.

Ghi chú

Để đơn giản, chỉ có một điều khiển được đề cập vào lúc này. Chỉ một nhóm các ô chọn được trình bày trên mẫu biểu. Không có lý do đặc biệt nào để nhóm chúng với nhau ngoài mục đích minh họa khái niệm khung là gì.

Củng cố

Đóng khung các đối tượng với nhau bằng điều khiển khung (Frame), bạn có thể khởi tạo các nhóm điều khiển làm việc cùng nhau như một mẫu biểu nhỏ trong cửa sổ Form lớn hơn.

DANH SÁCH COMBO XỔ XUỐNG

Khái niệm

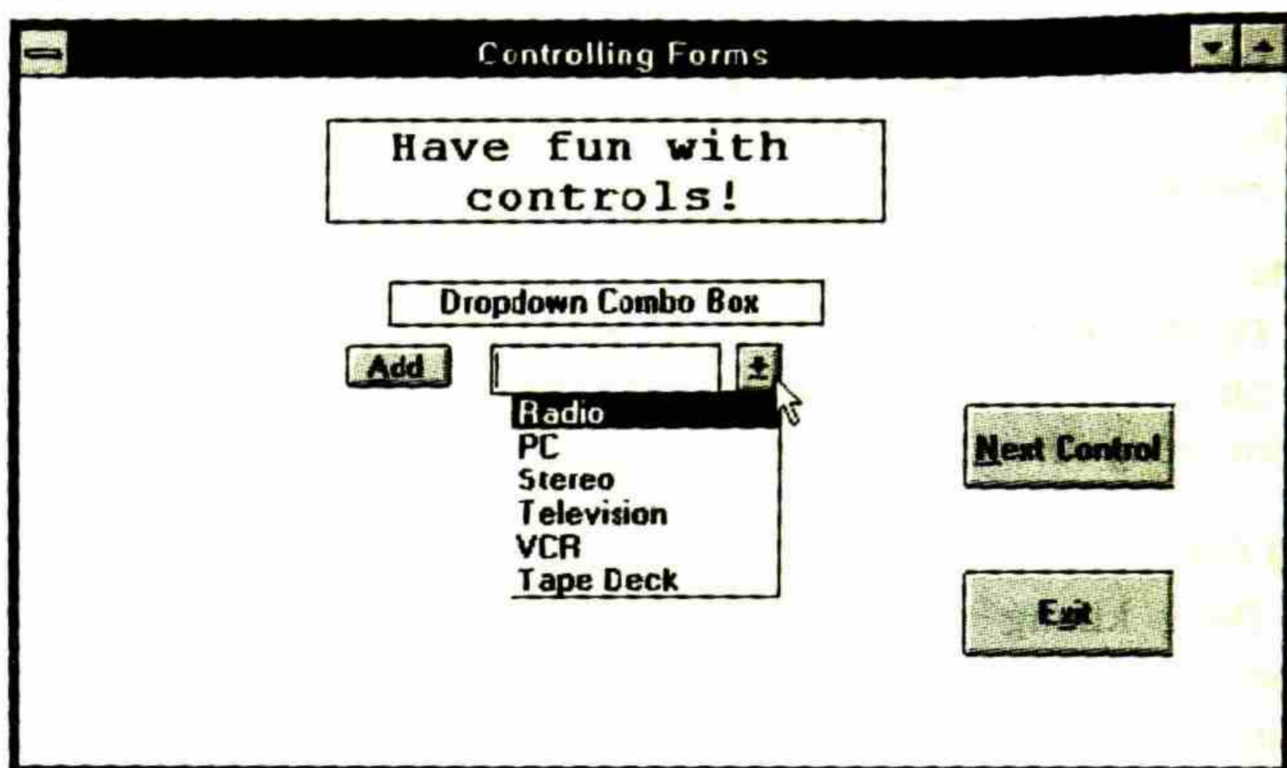
Danh sách combo xổ xuống là một trong ba loại danh sách bạn có thể cung cấp cho người dùng. Danh sách combo xổ xuống sẽ tiết kiệm không gian màn hình bằng cách chỉ chiếm một dòng trên mẫu biểu cho đến khi người dùng mở danh sách để hiển thị các mục còn lại trong nó.

Điều khiển hộp combo (Combo Box) thực tế chia làm hai loại, phụ thuộc vào cách khởi tạo hộp combo của bạn trên mẫu biểu. Hai loại hộp combo là:

- Hộp combo xổ xuống
- Hộp combo đơn giản

Nhấp Next Control để xem hộp combo xổ xuống trên màn hình. Thông thường trong trường hợp này, một nút lệnh xuất hiện kế hộp combo xổ xuống. (Hãy xem lại cách điều chỉnh kích cỡ cho từng điều khiển, kể cả nút lệnh, để những điều khiển chiếm vừa đủ không gian màn hình cần cho nó.)

Một danh sách các mục được lưu trong hộp combo xổ xuống này. Để xem danh sách, nhấp mũi tên xuống cuối hộp combo. Bạn sẽ thấy danh sách thiết bị điện tử như minh họa ở Hình 3.10. Nhấp lại mũi tên xuống một lần nữa, danh sách sẽ cuộn trở lại và xóa sạch sự lộn xộn của dữ liệu phụ trên màn hình.



Hình 3.10. *Sau khi mở điều khiển hộp combo xổ xuống.*

Mách nước

Lúc không gian màn hình trở nên quý giá, hãy sử dụng điều khiển hộp combo xổ xuống khi bạn phải cung cấp một danh sách các mục cho người dùng. Người dùng có thể hiển thị danh sách khi cần để xem những gì trong danh sách và họ có thể phục hồi danh sách trở lại tình trạng bình thường khi công việc hoàn tất.

Vùng trống ở đầu danh sách để cho người dùng thêm các mục vào hộp combo xổ xuống. Bạn nên thêm một nút lệnh cạnh hộp combo, như trong ứng dụng CONTROLS.VBP, để người dùng có thể bổ sung các mục mới sau khi nhập chúng vao vùng trống. Hãy thêm hai mục bổ sung bằng cách thực hiện các bước sau:

1. Nhập từ **Disc Player**.
2. Nhấp nút Add. Hộp nội dung dữ liệu nơi bạn đã nhập dữ liệu mới sẽ trống, nhưng Visual Basic đã nhập Disc Player vào danh sách.
3. Nhập tiếp từ **Amplifier** và nhấp nút Add.
4. Hiển thị danh sách xổ xuống bằng cách nhấp mũi tên xuống. Hãy chú ý rằng Visual Basic đã thêm hai mục mới vào cuối danh sách.

Ghi chú

Sau này, bạn sẽ học cách Visual Basic sắp xếp các mục theo thứ tự abc – Bạn không phải viết bất kỳ mã lệnh nào để sắp xếp các mục.

Các mục tồn tại trong danh sách hộp combo xổ xuống cho đến khi bạn thoát khỏi chương trình. Phần sau, bạn sẽ học cách lưu các mục trong danh sách và khởi tạo những giá trị cho danh sách bằng mã lệnh.

Củng cố

Điều khiển hộp combo xổ xuống giúp người dùng xem và thêm các mục vào danh sách một cách thuận tiện. Ưu điểm của hộp combo xổ xuống là người dùng có thể xem các mục trong danh sách mà không mất nhiều không gian màn hình như các điều khiển khác. Người dùng có thể hiển thị toàn bộ danh sách bằng cách nhấp mũi tên xuống.

HỘP COMBO ĐƠN GIẢN

Khái niệm

Điều khiển hộp combo đơn giản giống hệt hộp combo xổ xuống, ngoại trừ việc nó được hiển thị ở dạng xổ xuống. Nói cách khác, nếu không bị hạn chế về không gian màn hình, có lẽ bạn muốn dùng hộp combo đơn giản để hiển thị và chọn các giá trị từ một danh sách mà không cần nhấn phím phụ để mở danh sách đó.

Hãy nhấp tiếp nút lệnh Next Control để xem hộp combo đơn giản. Như bạn sẽ thấy, không có điểm khác biệt nào giữa hộp combo đơn giản với hộp combo xổ xuống ngoại trừ việc hộp combo đơn giản luôn luôn mở.

Nếu hộp combo đơn giản không đủ dài để hiển thị tất cả các mục trong danh sách, Visual Basic sẽ đưa thanh cuốn dọc vào danh sách để người dùng có thể cuộn danh sách và xem các mục. Như với hộp combo xổ xuống, người dùng có thể thêm các mục vào hộp combo đơn giản bằng cách nhập giá trị mới và nhấp nút lệnh bạn khởi tạo.

Nhập giá trị Rugby và nhấp nút lệnh Add. Cuộn tới cuối danh sách – bạn có thể sử dụng phím Page Down để cuộn – xem tên hình thức thể thao mới.

Ghi chú

Còn một loại hộp combo thứ ba, được gọi là hộp danh sách xổ xuống, sẽ không đề cập ở đây. Hộp danh sách xổ xuống chẳng cung cấp điều gì mới và bạn có thể thay thế nó bằng cách sử dụng điều khiển hộp combo hay hộp danh sách khác. Việc đề cập đến nó chỉ thêm phức tạp công việc mà thôi.

Củng cố

Điều khiển hộp combo giúp hiển thị và tập hợp các danh sách giá trị dễ dàng hơn. Người dùng không cần biết cách mở hộp combo đơn giản – nó luôn luôn mở.

HỘP DANH SÁCH VỚI QUÁ TRÌNH LỰA CHỌN

Khái niệm

Các hộp danh sách dễ hiểu hơn so với hai điều khiển hộp combo. Nếu bạn muốn người dùng chọn một mục trong danh sách và không cho họ thêm các mục mới vào danh sách, hãy sử dụng hộp danh sách.



Hình 3.11. *Chọn từ hộp danh sách.*

Hình 3.11 mô tả màn hình sẽ hiển thị khi bạn nhấp lại nút lệnh Next Control. Sau này bạn sẽ học cách nhập một hay nhiều mục vào hộp danh sách để người dùng chọn.

Lưu ý

Ô chọn và nút tùy chọn cho người dùng cơ hội chọn lựa, nhưng bạn không nên dùng ô chọn hay nút tùy chọn khi có nhiều lựa chọn muốn chọn. Quá trình cuộn trong điều khiển hộp danh sách và hộp combo sẽ sử dụng không gian màn hình hiệu quả hơn.

Nhấp mouse vào một mục trong hộp danh sách, Visual Basic sẽ hiện sáng mục đó. Thông qua quá trình lập trình, bạn sẽ biết khi nào người dùng chọn một mục, và bạn có thể phân tích những gì họ chọn. Nhấp mục khác để chọn nội dung khác.

Nếu ứng dụng đòi hỏi nhiều lựa chọn, bạn có thể khởi tạo hộp danh sách có thể chọn được nhiều mục. Điều khiển hộp danh sách trong ứng dụng CONTROLS.VBP cho phép nhiều lựa chọn. Nhấn–giữ phím Ctrl, Alt, hoặc Shift và nhấp một hay hai mục bổ sung. Chú ý rằng Visual Basic hiện sáng tất cả các mục. Bạn có thể quyết định người dùng có thể chọn chỉ một hay nhiều mục trong hộp danh sách. Cũng như trong hộp combo, bạn có thể yêu cầu Visual Basic sắp xếp các mục trong hộp danh sách theo thứ tự abc.

Nếu muốn, bạn có thể xem lại tất cả điều khiển bằng cách nhấp nút lệnh Next Control. Bây giờ bạn sẽ thấy tất cả điều khiển đã cung cấp trong ứng dụng CONTROLS.VBP. Mặc dù bạn sẽ học nhiều điều khiển hơn trong các bài sau, nhưng bài này đã mô tả các điều khiển chính mà tất cả lập trình viên Visual Basic cần làm chủ. Để bạn nắm chắc tất cả điều khiển, xem Hình 3.1 xác định vị trí các điều khiển trên cửa sổ Toolbox.

Cảnh báo

Không phải bất kỳ chương trình Windows nào bạn viết sẽ chứa đủ các điều khiển này. Thực ra, không một chương trình Windows nào lại cần đến tất cả điều khiển này. Chương trình sẽ trở nên lộn xộn và đòi hỏi người dùng quá nhiều loại trả lời khác nhau.

Củng cố

Hộp danh sách là điều khiển cung cấp cho người dùng các tùy chọn mà họ có thể chọn. Trình ứng dụng của bạn sẽ chịu trách nhiệm khởi tạo các giá trị cho hộp danh sách. Người dùng không thể thêm các mục vào một hộp danh sách.

Bài tập

Kiến thức tổng quát

1. Trước khi chạy trình ứng dụng từ trong môi trường Visual Basic, bạn phải làm gì?
2. Loại tập tin nào lưu nội dung mô tả trình ứng dụng?
3. Phần mở rộng của các tập tin project là gì?
4. Bạn có thể tìm thấy nội dung của một tập tin project ở đâu?
5. Công dụng của điều khiển?
6. Bạn lấy điều khiển ở đâu để thiết kế trình ứng dụng của mình?
7. Tại sao lại phải thêm nút lệnh Exit vào các trình ứng dụng?
8. Đúng hay Sai: Người dùng có thể thay đổi văn bản trong điều khiển nhãn (Label).
9. Điều khiển nào cần sử dụng khi muốn nhận văn bản từ người dùng?
10. Để văn bản hiển thị trong điều khiển nhãn hay hộp nhập, bạn phải làm gì?
11. Các điều khiển văn bản hoạt động giống trình xử lý từ như thế nào?
12. Điều khiển giúp người dùng truy xuất các nút nhấn gọi là gì?
13. Đúng hay Sai: Người dùng có thể chọn nhiều nhất một ô chọn từ danh sách điều khiển ô chọn.
14. Đúng hay Sai: Người dùng có thể chọn nhiều nhất một tùy chọn từ danh sách các nút tùy chọn.
15. Điều gì sẽ xảy ra nếu người dùng chọn lại ô chọn đã chọn?
16. Điều khiển nào cho phép bạn thêm nhiều nhóm nút tùy chọn vào mẫu biểu?
17. Đúng hay Sai: Người dùng có thể thêm các hạng mục vào danh sách hiển thị trong điều khiển hộp combo xổ xuống.

18. Đúng hay Sai: Người dùng có thể thêm các hạng mục vào danh sách hiển thị trong điều khiển hộp combo đơn giản.
19. Đúng hay Sai: Người dùng có thể thêm các hạng mục vào danh sách hiển thị trong hộp điều khiển danh sách.
20. Đúng hay Sai: Người dùng có thể chọn nhiều hơn một hạng mục trong hộp danh sách.
21. Điều gì dùng làm nền trình ứng dụng và chứa tất cả các điều khiển người dùng sẽ làm việc?
22. Đúng hay Sai: Visual Basic cho bạn, một lập trình viên, cơ hội bổ sung danh sách trong điều khiển hộp danh sách và hộp combo đồng thời hiển thị danh sách theo thứ bậc abc.
23. Hãy cho biết ít nhất hai cách người dùng có thể kích hoạt nút lệnh.
24. Tại sao nút lệnh thường tiếp sau điều khiển hộp combo?

Tìm lỗi kỹ thuật

25. An Huy đang viết trình ứng dụng Visual Basic hiển thị thông tin hóa đơn khách hàng. Vì số lượng thông tin của mỗi khách hàng khá lớn, nên An Huy thấy màn hình lấy thông tin là quá chật chội. An Huy thêm danh sách hàng hóa mà khách hàng đã mua vào điều khiển hộp combo đơn giản. Bằng cách đó, cô thư ký có thể cuộn qua danh sách các mặt hàng và mặt hàng mua thêm vào danh sách họ chọn. Điều khiển hộp combo đơn giản giúp tiết kiệm khá nhiều không gian màn hình. Bạn sẽ khuyên An Huy thế nào?

Phần nâng cao

Giả sử bạn đang viết trình ứng dụng cung cấp cho người dùng ba giá trị để họ chọn nhiều nhất một giá trị. Điều khiển nào là thích hợp nhất?

Bài 4

Điều khiển và thuộc tính

- **Môi trường hướng biến cố**
- **Thuộc tính điều khiển**
- **Qui ước đặt tên**
- **Thiết kế nhanh trình ứng dụng**

Trong bài này, bạn sẽ thiết kế chương trình bằng Visual Basic. Bạn sẽ viết chương trình Windows hầu hiển thị cửa sổ có khả năng hiệu chỉnh kích cỡ, nút lệnh, nút điều khiển cùng menu.

Tuy nhiên, trước khi tạo trình ứng dụng đầu tiên, bạn phải tìm hiểu cách thức chương trình Visual Basic tương tác với môi trường Windows.

MÔI TRƯỜNG HƯỚNG BIẾN CỐ

Khái niệm

Chương trình Windows vận hành khác hẳn chương trình chạy trên DOS. Thay vì chương trình điều khiển người dùng, thì giờ đây người dùng sẽ điều khiển chương trình. Khi người dùng kích hoạt lệnh menu hay điều khiển, Windows sẽ tạo biến cố mô tả các hành động riêng lẻ.

Hãy hình dung bạn đang lái xe trên đường một mình. Tất cả các điều khiển đều do bạn thực hiện – lái xe, đèn xi-nhan, bật đèn pha, thắng, dừng, đổi hướng, bấm còi, bật máy lạnh, v.v. Bất kỳ lúc nào bạn cũng có thể hãm phanh, rẽ trái, nghe radio, điều chỉnh kính chiếu hậu hay tốc độ. Điều kiện trong xe lẫn tình huống thực tế định rõ công việc kế tiếp bạn sẽ thực hiện.

Khi nhấn ga, chân của bạn có làm xe hơi chạy nhanh hơn không? Câu trả lời là không. Bạn cần có rất nhiều động tác để có thể tăng tốc độ lên đến 50 dặm một giờ. Việc đạp pedal tạo ra một biến cố. Đó là thêm ga cho buồng đốt và làm xe hơi chạy nhanh hơn. Xe hơi của bạn

thực hiện công việc của nó và thao tác, điều chỉnh dựa trên biến cố mà bạn kích hoạt.

Khái niệm mới

Biến cố có thể là quá trình di chuyển mouse, nhấp mouse, nhấn phím, hay đáp ứng điều khiển.

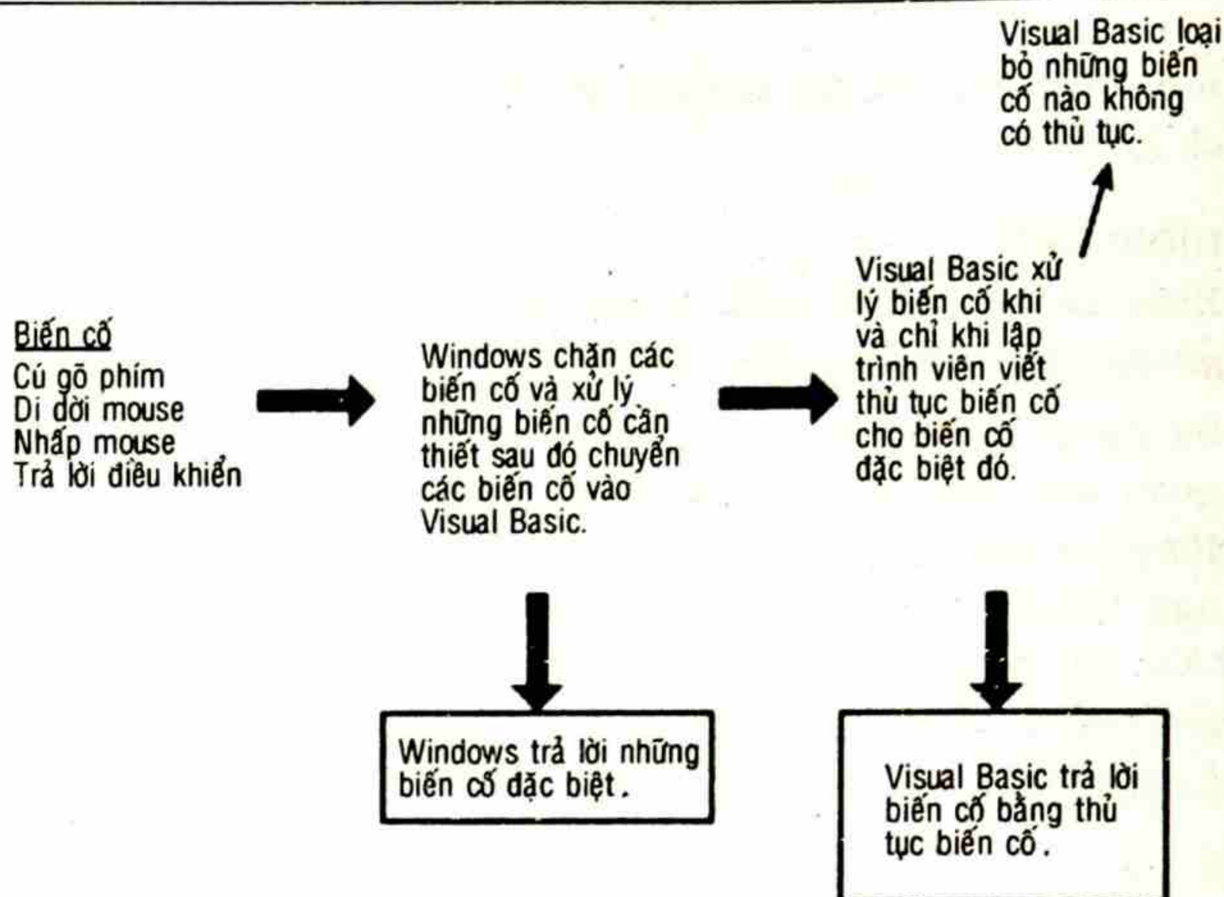
Sử dụng chương trình Windows giống như lái xe hơi theo mối tương quan sau: bạn không thực hiện cùng một lúc nhiều thao tác và hành động mà bạn thực hiện sẽ làm cho biến cố xác định xảy ra. Trong thuật ngữ Windows, *biến cố* (event) *là hành động* người dùng hiểu rõ. Bất cứ khi nào người dùng nhấp mouse, nhấn phím, đáp ứng điều khiển hay chọn menu, biến cố sẽ xảy ra. Windows giám sát suốt quá trình chạy chương trình Visual Basic và tìm kiếm biến cố.

Ghi chú

Quá nhiều điều có thể diễn ra trong chương trình dựa trên GUI. Nếu bạn chạy chương trình kế toán trên DOS, chương trình thích hợp hơn sẽ giới thiệu cho bạn menu lựa chọn được giới hạn. Ngay sau khi chọn từ menu, chương trình mới dẫn dắt bạn qua bước kế tiếp. Trong chương trình Windows, bạn được phép lựa chọn điều khiển và có thể trả lời bất kỳ điều khiển nào theo thứ tự bất kỳ. Chương trình phải có khả năng phán đoán khi nào biến cố xảy ra và xử lý nó.

Windows và Visual Basic kiểm tra liên tục quá trình vận hành chương trình. Khi người dùng nhấp nút lệnh làm xuất hiện một biến cố bất kỳ, Windows sẽ lưu giữ biến cố và gửi nó cho Visual Basic.

Visual Basic không đáp ứng biến cố hệ thống đặc biệt khi người dùng nhấn tổ hợp phím Alt+Tab hầu chuyển đến trình ứng dụng khác đang chạy trong bộ nhớ. Điều đó giải thích tại sao Windows phải lưu giữ tất cả biến cố và chuyển biến cố có thể xử lý cho Visual Basic. Hình 4.1 trình bày mối quan hệ giữa biến cố và thủ tục biến cố. Lưu ý Windows luôn lưu giữ biến cố và chuyển biến cố thích hợp đến chương trình Visual Basic, tại thời điểm đó Visual Basic sẽ đáp ứng biến cố nếu thủ tục có tác dụng.



Hình 4.1. *Windows lưu giữ biến cố trước khi chuyển nó cho Visual Basic.*

Khái niệm mới

Thủ tục biến cố sẽ đáp ứng biến cố.

Khi Visual Basic nhận biến cố từ Windows, nó kiểm tra xem lập trình viên đã viết thủ tục cho biến cố chưa. Giả như thủ tục biến cố có sẵn, Visual Basic sẽ thi hành biến cố. Trong chương trình CONTROLS.VBP vốn đã chạy ở bài học trước, nhấp nút lệnh Next Control tạo điều khiển kế tiếp trong tập hợp điều khiển chương trình sẽ hiển thị. Cách duy nhất chương trình có thể trả lời nút lệnh Next Control là viết thủ tục cho biến cố nghĩa là biến cố nhấn nút lệnh.

Chương trình có thể bao gồm các điều khiển và vẫn không đáp ứng mọi biến cố. Ví dụ, trong CONTROLS.VBP, khi bạn nhấp vào ô chọn, không có điều gì xảy ra ngoại trừ các ô chọn đó được chọn hay không được chọn. Điều khiển tiến hành kiểm tra hay không kiểm tra ô chọn vốn không có thủ tục biến cố nào cần để làm điều đó. Mặc dù, một khi bạn chọn các ô chọn, chương trình tuyệt đối không hề đả động gì đến những ô chọn được chọn bởi vì không có thủ tục biến cố hoạt động bên trong CONTROLS.VBP khi người dùng nhấp chọn một ô chọn. Các ô chọn khả dụng trong chương trình đó chỉ nhằm minh họa cách chúng thi hành.

Cảnh báo

Đa số, nhưng không phải tất cả, các biến cố được người dùng kích hoạt. Một số biến cố Windows bên trong có thể dùng để kích hoạt các hành động. Bên cạnh đó, bạn có thể chỉ thị cho chương trình trả lời sau một khoảng thời gian đã định, như đếm số giây chẳng hạn. Trong trường hợp này, khoảng thời gian được xem là biến cố.

Có phải đây là kỹ thuật về biến cố và các thủ tục biến cố? Thật ra, quá trình bắt biến cố và thủ tục biến cố rất dễ dàng. Môi trường Visual Basic được khởi tạo để tạo các thủ tục biến cố cho bạn khi bạn yêu cầu chúng. Cuối cùng khi muốn chương trình Windows của bạn đáp ứng biến cố, phải chắc chắn rằng bạn đã viết thủ tục biến cố cho nó. Nếu bạn muốn bỏ qua biến cố xác định, đừng viết thủ tục biến cố đó.

Mách nước

Đa số biến cố bạn xử lý đã quá rõ ràng. Ví dụ, bạn thêm nút lệnh vào mẫu biểu, bạn muốn làm một điều gì đó khi người dùng nhấp nút lệnh. Tuy nhiên, nếu người dùng tìm cách kéo nút lệnh, có lẽ bạn muốn bỏ qua biến cố này bởi vì người dùng không nên tự quyền di chuyển điều khiển trong quá trình thi hành chương trình.

Củng cố

Hầu hết mọi việc có thể xảy ra trong quá trình thi hành chương trình Windows, đó là biến cố. Một khi bạn thiết kế mẫu biểu chương trình Windows và đặt các điều khiển trên nó, bạn phải viết mã lệnh đáp ứng biến cố. Mã lệnh đó được xây dựng trong các thủ tục biến cố. Bằng cách này, mỗi thủ tục biến cố giống như một chương trình thu nhỏ mà bạn viết cho từng điều khiển có biến cố gây nên hành động cần gọi chương trình.

THUỘC TÍNH ĐIỀU KHIỂN

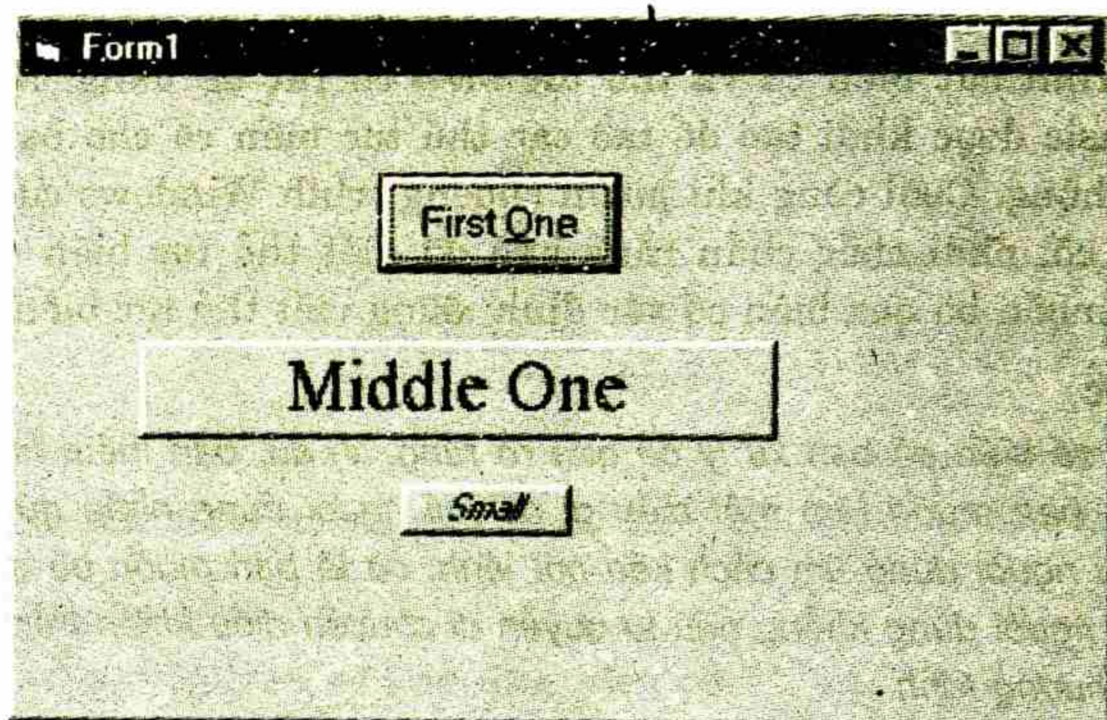
Khái niệm

Điều khiển khác loại sẽ làm việc khác nhau. Bạn có thể sử dụng điều khiển cho nhiều mục đích. Tuy nhiên, ngay cả giữa các điều khiển cùng loại vẫn tồn tại điểm khác biệt. Sự khác biệt này tập trung vào thuộc tính điều khiển.

Khái niệm mới

Thuộc tính phân biệt điều khiển này với điều khiển khác.

Hãy xem ba điều khiển trong Hình 4.2. Cả ba điều khiển đều là nút lệnh nhưng chúng trông khác nhau. Người dùng có thể nhấp bất kỳ nút lệnh nào, mỗi nút lệnh có một tập hợp các thuộc tính khác nhau.



Hình 4.2. *Ba nút lệnh với các thuộc tính khác nhau.*

Nút lệnh đầu có kích thước như thường thấy trong chương trình Windows. Visual Basic ấn định tự động kích thước này khi bạn thêm nút lệnh vào trình ứng dụng trừ khi bạn xác định điều gì khác. Nút lệnh thứ hai rộng hơn nút lệnh đầu, và không có phím truy xuất. Đề mục của nó không được hiển thị ở dạng chữ đậm, bên cạnh đó phông chữ lại không chuẩn. Nút lệnh thứ ba rất nhỏ, phụ đề của nó có dạng chữ nghiêng. Bởi vì dạng chữ nghiêng nhỏ trên nút lệnh rất khó đọc, vì thế nên cẩn thận khi sử dụng chúng với các điều khiển nhỏ.

Mách nước

Nhỏ thường tốt hơn lớn.

Không nên thêm quá nhiều điều khiển. Trình ứng dụng thường đòi hỏi các nút lệnh khác nhau về kích thước mặc định, nhưng càng giống nhau càng tốt. Việc đặt nhiều phông chữ trên các nút lệnh trong cùng mẫu biểu ít khi được hoan nghênh.

Một trong những bước quan trọng nhất bạn cần thực hiện khi viết chương trình Visual Basic là định các thuộc tính điều khiển. Khi cuốn sách này chỉ cho bạn cách thêm điều khiển mới vào trình ứng dụng, bạn sẽ làm quen với tất cả thuộc tính của nó. Vì vậy, bài tiếp theo sẽ chỉ cho bạn cách đặt nút lệnh. Trước khi tìm hiểu cách sắp xếp, hãy tìm hiểu các thuộc tính của nút lệnh.

Ghi chú

Năm thì mới họa bạn mới thay đổi tất cả thuộc tính của điều khiển khi đặt nó trên mẫu biểu. Tuy nhiên, bạn sẽ thấy mỗi thuộc tính đều có giá trị của nó. Khi học điều khiển mới, bạn sẽ biết những gì có thể và không thể thực hiện.

Ngay cả mẫu biểu cũng có các thuộc tính: lập trình viên Visual Basic sử dụng thuật ngữ chung – *đối tượng* (object) – để chỉ các điều khiển đặt trên mẫu biểu. Thực tế, ngay cả mẫu biểu mẫu cũng là đối tượng. Từ đối tượng nghĩa là những công việc khác nhau trong ngôn ngữ máy tính. Một đối tượng Visual Basic có ít việc để làm hơn so với các đối tượng trong ngôn ngữ lập trình hướng đối tượng như C++ chẳng hạn.

Mỗi đối tượng trong chương trình Visual Basic đều có các thuộc tính. Ngay cả mẫu biểu cũng có thuộc tính. Qua Hình 4.2, bạn có thể thêm đề mục mô tả mẫu biểu. Tiêu đề của mẫu biểu là một trong số những thiết đặt thuộc tính của nó. Màu nền cũng là một thuộc tính khác mà bạn có thể ấn định. Tuy nhiên, bạn nên định màu trắng cho nền mẫu biểu, nó được dùng trong hầu hết chương trình Windows mà lập trình viên có thể thay đổi.

Mách nước

Bạn có thể thêm thuộc tính cho điều khiển ở đâu? Trong cửa sổ Properties. (Một câu trả lời hay, có phải không?)

Củng cố

Mỗi điều khiển có một tập hợp giá trị thuộc tính khác nhau. Bạn thường định giá trị thuộc tính ban đầu cho các điều khiển trên mẫu biểu. Trong suốt quá trình thi hành chương trình, mã lệnh của bạn thường thay đổi các giá trị thuộc tính. Ví dụ, ứng dụng CONTROLS.VBP sẽ thay đổi thuộc tính đề mục của nút lệnh từ **Press Me** thành **Once**

Again, rồi đổi lại **Press Me**. Ban đầu, bạn đã thêm đề mục **Press Me** khi thêm nút lệnh vào mẫu biểu. Sau đó, bạn sử dụng mã lệnh thay đổi đề mục trong quá trình thi hành chương trình. Điều gì giúp Visual Basic biết để thay đổi đề mục? Khi người dùng nhấn nút lệnh, biến cố nhấp nút lệnh sẽ xảy ra. Lúc đó thủ tục biến cố được viết cho biến cố riêng biệt đó sẽ thay đổi đề mục. Hãy đọc tiếp và bạn sẽ thấy rằng việc tạo các thủ tục biến cố là không có gì khó khăn và phức tạp như ban đầu bạn nghĩ.

QUI ƯỚC ĐẶT TÊN

Khái niệm

Bạn sẽ tìm hiểu tất cả thuộc tính điều khiển trong phần sau, còn bây giờ thì hãy làm quen với một thuộc tính điều khiển quan trọng là thuộc tính Name. Tất cả điều khiển đều có thuộc tính Name. Thuộc tính Name là tên của từng điều khiển riêng biệt. Không có một tên duy nhất, bạn sẽ khó lòng phân biệt điều khiển này với điều khiển khác trong mã lệnh Visual Basic. Mặc dù Visual Basic gán các tên mặc định tới tất cả điều khiển, nhưng vẫn phải nắm những qui ước thay đổi tên này để bạn có thể nhớ chúng dễ dàng hơn khi thêm chúng vào chương trình.

Khái niệm mới

Qui ước đặt tên là tập hợp các qui tắc đặt tên.

Thuộc tính đầu tiên bạn nên thiết đặt là thuộc tính Name. Bài này không chỉ cho bạn cách đặt thuộc tính Name, vì điều đó sẽ được trình bày trong bài kế tiếp. Tuy nhiên, bạn phải biết rằng lập trình viên Visual Basic không gán tên tùy tiện cho các điều khiển. Lập trình viên muốn tạo cho công việc lập trình của họ ít căng thẳng hơn bằng cách đưa ra qui ước đặt tên khi họ chọn tên cho điều khiển của mình.

Qui ước không chỉ do một nhóm người tập hợp lại và là một tập hợp các qui tắc tiêu chuẩn để bạn đi theo. Ví dụ, bạn đã làm quen với qui ước đặt tên tập tin được sử dụng trong Windows và DOS (không quá 8 ký tự cho tên và không quá 3 ký tự cho phần mở rộng). Trong Visual Basic, qui ước đặt tên cho điều khiển cũng đơn giản. Khi bạn quyết định tên cho điều khiển, phải đảm bảo các điều kiện sau:

- Tên có thể từ 1 đến 40 ký tự.
- Tên phải bắt đầu với ký tự chữ, có thể là chữ hoa hay chữ thường.
- Sau ký tự đầu tiên, tên có thể chứa ký tự, số hay dấu gạch dưới.

Khái niệm mới

***Từ khóa (reserved word)** là một lệnh Visual Basic.*

1. Tên không thể trùng với từ khóa.
2. Tên phải gợi nhớ. Lấy ví dụ, mặc dù bạn có thể đặt tên nút lệnh thoát đang tồn tại là Rose, nhưng cmdExit thì tốt hơn; đó là qui ước và dễ nhớ hơn.

   Các tên sau đây là hợp lệ:

   cmdExit

   ListBoxJan

   Nov95Combo

   TitleScreen

   Và những tên sau không hợp lệ:

   Select

   723

   cmdExit&Leave

   Select là lệnh Visual Basic. 723 không bắt đầu với bằng ký tự. cmdExit&Leave có chứa ký tự không hợp lệ &.

Lưu ý

Dù dấu gạch dưới là ký tự hợp lệ, song nhiều lập trình viên không thích sử dụng nó bởi vì đôi khi nó trông giống dấu trừ.

Hãy nắm bắt qui ước đặt tên này. Nó không chỉ áp dụng với thuộc tính Name mà còn với các thành phần khác trong Visual Basic như biến và thủ tục.

Lập trình một lần, bảo trì thường xuyên: Hiếm khi bạn hoàn tất chương trình sau khi viết nó. Bạn thường phải nâng cấp chương trình để làm mới những thay đổi trong môi trường bạn đang sử dụng. Khi cập nhật chương trình đã viết, bạn đang làm công việc gọi là bảo trì chương trình.

Ví dụ, nếu đã viết chương trình kế toán cho một công ty nhỏ liên kết với công ty thứ hai, phòng kế toán có lẽ phải giữ tình trạng kế toán riêng biệt cho cả hai công ty cho đến khi kết thúc năm kế toán hiện hành. Bạn phải sửa đổi chương trình để nó phân biệt dữ liệu giữa hai công ty và lưu trữ thành hai bộ dữ liệu riêng. Bạn phải mất thêm thời gian đưa ra những tên có ý nghĩa cho điều khiển, và một ít thời gian tưởng tượng những gì từng điều khiển sẽ thực hiện. Mọi việc sẽ rõ ràng hơn nếu điều khiển có tên cmdComputeProfit được kích hoạt để thực hiện tính lãi lỗ.

Ở đây mạnh dạn đề nghị với bạn rằng nên chấp nhập các tiêu chuẩn đặt tên bắt đầu bằng tiếp đầu ngữ của những điều khiển được liệt kê trong Bảng 4.1. Bảng 4.1 chỉ ra ba ký tự đầu mà bạn nên thêm vào trước tên điều khiển. Tiếp đầu ngữ cho biết đó là loại điều khiển gì. Vì vậy, từ tên của nó bạn có thể biết loại điều khiển đang làm việc. Dĩ nhiên, nếu bạn đang đặt điều khiển trên mẫu biểu và đặt tên cho nó, bạn sẽ biết nó thuộc loại điều khiển nào. Tuy nhiên, khi thêm mã lệnh vào các thủ tục biến cố sau đó, ba ký tự đầu của tên điều khiển sẽ giúp bạn nắm rõ loại điều khiển và ngăn bạn khỏi phải viết thủ tục biến cố sai cho điều khiển không có biến cố riêng biệt đó.

Ghi chú

Bảng 4.1 liệt kê tất cả qui ước đặt tên của điều khiển và mẫu biểu, kể cả những điều khiển chưa được giới thiệu.

Bảng 4.1. *Các tiêu chuẩn đặt tên tiền tố cho điều khiển*

Tên tiền tố	Điều khiển
cbo	Hộp Combo
chk	Ô chọn (Check Box)
cmd	Nút lệnh (Command Button)
dir	Hộp danh sách thư mục (Directory List Box)
drv	Hộp danh sách ổ đĩa (Drive List Box)
fil	Hộp danh sách tập tin (File List Box)
fra	Khung (Frame)
frm	Mẫu biểu (Form)
grd	Khung lưới (Grid)

hsb	Thanh cuốn ngang (Horizontal Scroll Bar)
img	Hình ảnh (Image)
lbl	Nhãn (Label)
lin	Đường kẻ (Line)
lst	Hộp danh sách (List Box)
mnu	Menu
ole	OLE client
opt	Nút tùy chọn (Option Button)
pic	Hộp Picture
shp	Hình dạng (Shape)
tmr	Bộ định giờ (Timer)
txt	Hộp nhập (Text Box)
vsb	Thanh cuốn dọc (Vertical Scroll Bar)

Sau đây là một số tên điều khiển sử dụng ba ký tự đầu từ Bảng 4.1:

frmOpening

lstSelections

chkBooksInPrint

Khi người dùng chuyển từ DOS lên Windows, họ luôn phàn nàn rằng Windows khác DOS nhiều, khó học và khó sử dụng. Ngay cả khi bạn chấp nhận Windows khó học và khó sử dụng, bạn cũng phải tìm hiểu qua giao diện Windows một lần. Điều này rất có ý nghĩa vì khi bạn nắm rõ một chương trình Windows, như môi trường lập trình Visual Basic hay Microsoft Excel, các chương trình Windows khác cũng sẽ hoạt động như vậy mà thôi. Hầu hết chương trình Windows đều có lệnh Exit trên menu File và hiển thị hộp thoại File Open như nhau, cũng như đều hiển thị màn hình màu trắng. Các nhà phát triển phần mềm không nhất thiết phải theo các chuẩn này, nhưng người dùng thích học các chương trình có dạng tương tự như vậy. Cẩm nang thiết kế trình ứng dụng sẽ giúp bạn nắm rõ hơn về các chuẩn này.

Củng cố

Khi đặt tên điều khiển, đừng gán tên tùy tiện hay trung thành với tên Visual Basic gán sẵn. Bạn sẽ không thích những tên đó. Hãy nghĩ về một tên ám chỉ mục đích của điều khiển, và sử dụng ba ký tự tiền tố như đã mô tả trong Bảng 4.1. Bằng cách theo tiêu chuẩn như thế, bạn sẽ bảo đảm chương trình của mình dễ bảo trì hơn sau này.

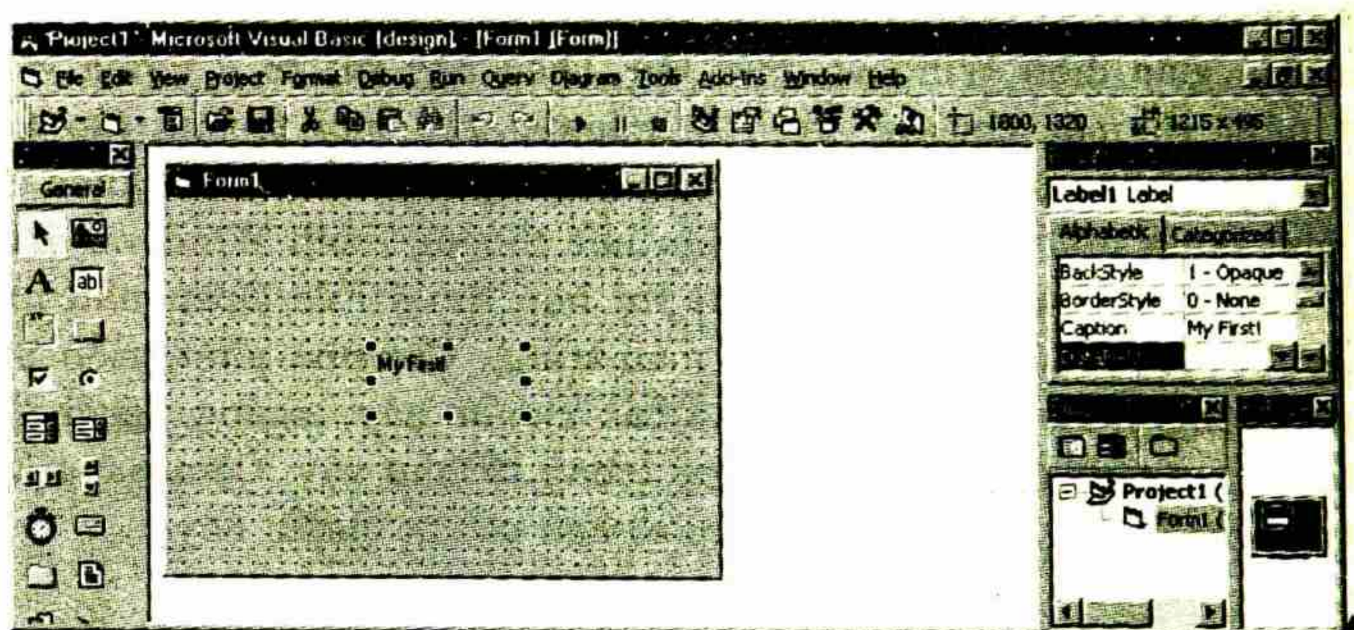
THIẾT KẾ NHANH TRÌNH ỨNG DỤNG

Khái niệm

Bây giờ bạn sẽ sử dụng Visual Basic để tạo một chương trình Windows. Nếu nghĩ rằng bạn chưa biết đầy đủ về Visual Basic để lập trình, cứ việc học hỏi bởi vì bạn sẽ tìm thấy cách thiết kế chương trình rất dễ dàng. Hãy thực hiện các bước sau để tạo trình ứng dụng Visual Basic đầu tiên của bạn:

Chọn File ➥ New Project để mở một cửa sổ Project mới. Visual Basic sẽ mở Project mới và cửa sổ Form trống để bạn có thể đặt vào đó các điều khiển.

1. Nhấp đúp điều khiển nhãn. Hãy nhớ rằng điều khiển nhãn là chữ A trên cửa sổ Toolbox. Một nhãn trống (với thuộc tính Name mặc định là Label1) sẽ hiển thị giữa cửa sổ Form.
2. Nhấn F4 để bật cửa sổ Properties.



Hình 4.3. *Sau khi thay đổi đề mục của nhãn.*

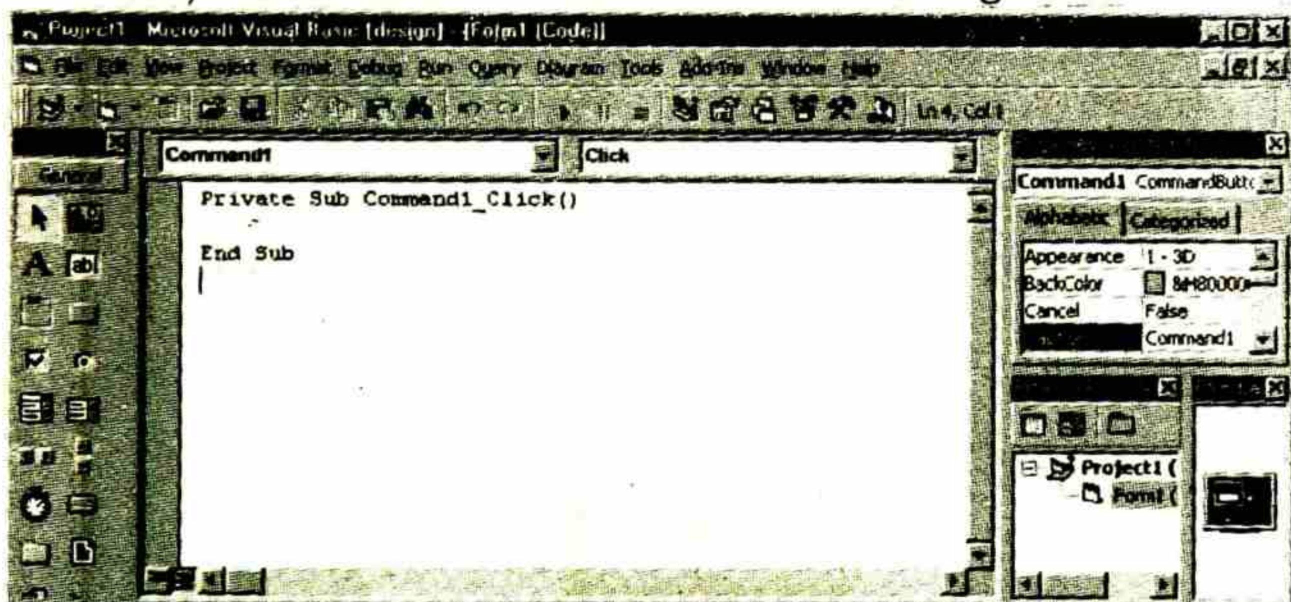
3. Cuộn các thuộc tính trong cửa sổ tới thuộc tính Caption. Thuộc tính Caption, mặc định chứa đề mục của nhãn. Bởi vì thuộc tính Caption lưu nội dung của nhãn, bạn nên thay đổi nội dung đó. Hãy hiện sáng thuộc tính Caption, nhập vào **My First!**, rồi nhấn Enter. Như mô tả trong Hình 4.3, nhãn sẽ hiển thị ngay lập tức đề mục mới.

4. Nhấn–giữ con trỏ mouse trên nhãn vừa thêm, kéo nhãn tới đầu cửa sổ Form. Góc trên của nhãn cách góc trên cửa sổ 1 inch. Canh giữa nhãn dưới tiêu đề mẫu biểu. Mặc định, tiêu đề là tên mẫu biểu, Form1. (Bạn sẽ thay đổi tên mặc định của điều khiển trong bài tiếp theo.)

5. Nhấp đúp điều khiển nút lệnh trên cửa sổ Toolbox. Một nút lệnh sẽ hiển thị ở giữa cửa sổ Form.

6. Nhập chính xác những gì sau đây: **E&xit**. Ký tự & gạch dưới chữ x, như bạn có thể thấy khi xem kết quả đề mục nút lệnh. Thuộc tính Caption là thuộc tính có thể thay đổi, cũng như thuộc tính cuối cùng bạn đã thay đổi khi làm việc với điều khiển nhãn là thuộc tính Caption của nó.

7. Canh giữa nút lệnh bằng cách kéo nút lệnh dịch sang trái một chút. Sau khi canh giữa nút lệnh, nhấp đúp nó. Ngay lập tức, Visual Basic mở cửa sổ Code như trong Hình 4.4.



Hình 4.4. *Cửa sổ Code sẽ mở khi bạn nhấp đúp điều khiển.*

8. Visual Basic biết rằng bạn muốn viết thủ tục biến cố sẽ thi hành bất cứ khi nào người dùng nhấp nút lệnh. Bạn có thể yêu cầu Visual Basic gọi thủ tục biến cố khi nhấp nút lệnh

bởi vì tên của thủ tục là Command1_Click(). (Đừng lo lắng về các dấu ngoặc ở bên phải.) Một thủ tục biến cố luôn có dạng sau đây: ControlName_EventName(). Tên mặc định của nút lệnh mới (cho đến khi bạn thay đổi thuộc tính Name) là Command1, và biến cố sẽ kích hoạt khi người dùng nhấp nút lệnh là Click. Vì vậy, mã lệnh xử lý biến cố nhấp mouse của nút lệnh là Command1_Click().

9. Hai dòng Visual Basic tự thêm vào thủ tục biến cố được gọi là các *dòng bao* (wrapper lines) bởi vì chúng bọc quanh mã lệnh bạn sẽ thêm vào thủ tục biến cố. Trong thủ tục biến cố này, bạn chỉ cần nhấn Tab và nhập lệnh **End**. Thủ tục biến cố trông giống như sau:

```
Sub Command1_Click ()
End
End Sub
```

10. Đó là tất cả. Để xem công trình bạn vừa thực hiện, đóng cửa sổ Code và nhấn phím F5 để chạy trình ứng dụng vừa tạo. Mẫu biểu mẫu của trình ứng dụng sẽ hiển thị trên màn hình với hai điều khiển của nó như Hình 4.5.



Hình 4.5. *Trình ứng dụng đầu tiên của bạn làm việc y như đùa.*

Để dừng trình ứng dụng và trở lại Visual Basic, hãy nhấp nút lệnh Exit. Ngay khi nhấp nút lệnh, thủ tục biến cố đã thêm vào sẽ thi hành. Câu lệnh End bạn thêm vào thủ tục biến cố sẽ thực hiện. Mục đích duy nhất của lệnh End là dừng trình ứng dụng.

Các biến cố nút lệnh khác: Để biết các biến cố khác có thể xảy ra với nút lệnh, hãy mở cửa sổ Code và nhìn tên các biến cố khác. Nhấp mũi tên xuống ở bên phải thủ tục cửa sổ Code, bạn sẽ thấy danh sách xổ xuống. Cuộn qua danh sách, bạn thấy rằng có biến cố DragDrop phát sinh khi người dùng kéo nút lệnh bằng mouse và biến cố KeyDown khi người dùng nhấn phím truy xuất nhanh của trình ứng dụng, ngoài ra còn nhiều biến cố khác.

Các loại điều khiển khác như hộp danh sách hay nhãn, có thể có biến cố y hệt biến cố của nút lệnh, nhưng những điều khiển khác nhau cũng có những biến cố khác nhau. Trong mục Object của cửa sổ Code: danh sách xổ xuống luôn chứa danh sách đối tượng hiện hành của trình ứng dụng như mẫu biểu và bất kỳ điều khiển nào bạn đã thêm vào mẫu biểu. Trong mục Proc là danh sách combo xổ xuống chứa tên các biến cố cho từng điều khiển đang hiện sáng trong danh sách Object.

Nếu muốn, bạn có thể cất giữ trình ứng dụng này. Khi đó, Visual Basic muốn bạn đặt tên cả mẫu biểu lẫn project. Thông thường, đặc biệt là các trình ứng dụng chỉ có một mẫu biểu, bạn đặt tên mẫu biểu cùng tên project, nhưng cả hai có phần mở rộng tên tập tin khác nhau. Giả sử bạn muốn lưu giữ project dưới tên MYFIRST, hãy thực hiện các bước sau:

1. Chọn File ➥ Save Project. Visual Basic mở hộp thoại Save As cho mẫu biểu. Nhập **MYFIRST** và nhấn Enter. Visual Basic thêm phần mở rộng là .FRM.
2. Kế tiếp Visual Basic sẽ hiển thị hộp thoại Save Project As. Nhập **MYFIRST** và nhấn Enter để lưu project. Visual Basic thêm phần mở rộng .VBP.
3. Bây giờ bạn có thể thoát Visual Basic và nghỉ ngơi.

Củng cố

Với một vài động tác nhấn phím và nhấp mouse, bạn có thể tạo một trình ứng dụng Visual Basic. Nếu chạy lại ứng dụng MYFIRST.VBP, bạn sẽ thấy rằng có thể phóng to, thu nhỏ hay định lại kích cỡ cửa sổ trình ứng dụng. Visual Basic tự động thêm nút điều khiển ở góc trái trên cửa sổ chương trình để bạn có thể điều khiển từ mức hệ thống Windows.

Bài tập

Kiến thức tổng quát

1. Biến cố là gì?
2. Tại sao môi trường lập trình GUI giống như hướng biến cố trong Windows?
3. Đúng hay Sai: Windows chuyển tất cả biến cố xảy ra trong chương trình của bạn vào Visual Basic.
4. Công dụng của thuộc tính?
5. Điều gì bạn phải viết để xử lý biến cố?
6. Tên hai thuộc tính của điều khiển bất kỳ bạn có thể nhớ trong bài này.
7. Đúng hay Sai: Mẫu biểu có các thuộc tính.
8. Đúng hay Sai: Mẫu biểu là đối tượng.
9. Cửa sổ nào bạn sử dụng để thay đổi các giá trị thuộc tính?
10. Bảo trì chương trình là gì?
11. Những thuận lợi của việc sử dụng tiền tố với 3 ký tự đã được đề nghị?
12. Tại sao bạn nên viết chương trình Windows trông giống và làm việc theo kiểu tương tự với các chương trình Windows khác?
13. Tên tập tin mô tả tất cả project mới là gì?
14. Đúng hay Sai: Visual Basic gán tên điều khiển mặc định là điều hay.

15. Bạn có thể mở cửa sổ Code cho thủ tục biến cố đầu tiên của điều khiển hay không?
16. Các thủ tục biến cố được đặt tên như thế nào?
17. Câu lệnh đầu tiên và cuối cùng trong thủ tục biến cố gọi là gì?
18. Thế nào là lệnh dừng chương trình đang chạy của Visual Basic?
19. Đối tượng của cửa sổ Code là gì, danh sách combo xổ xuống bao gồm những gì?
20. Tên tập tin của mẫu biểu khác với tên của một project như thế nào?
21. frmStartUp mô tả đối tượng nào?
22. cboNameChoice96 mô tả đối tượng nào?
23. cmdPrintIt mô tả đối tượng nào?

Tìm lỗi kỹ thuật

24. An Huy, một lập trình viên Visual Basic vừa có đĩa CD–ROM mới với số lượng phông chữ lớn. An Huy quyết định sử dụng một phông chữ cho một từ trong mẫu biểu của mình. Bạn có lời khuyên nhẹ nhàng nào giúp đỡ An Huy không?
25. Hãy mô tả những điểm sai trong từng thuộc tính Name sau:
    a. End
    b. 96JanSalesList
    c. cmdStar$
    d. July–List

Phần nâng cao

Hãy tạo project mới chứa hai nút điều khiển. Thêm câu lệnh Beep vào thủ tục biến cố Click của nút đầu tiên, và thay đổi thuộc tính Caption thành Ring a Bell. (Beep là lệnh phát tiếng kêu bíp trong máy PC.) Hãy thay đổi thuộc tính Caption của nút lệnh thứ hai thành Exit, và nhập lệnh End vào thủ tục biến cố Click của nó. Sau đó chạy chương trình và kiểm tra kết quả của bạn.

Bài thực hành 2

Chương trình Visual Basic

Tóm tắt

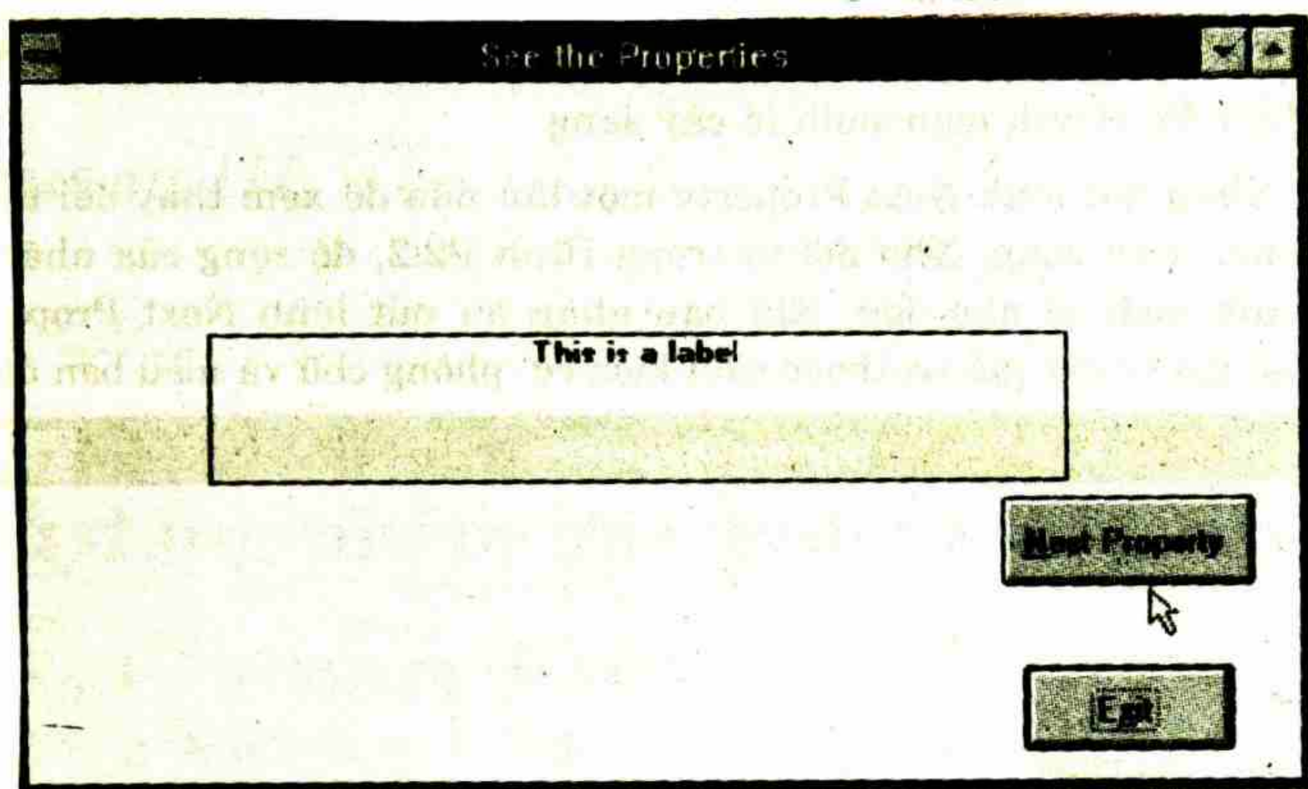
Chương này đã đề cập một số điều khiển và thuộc tính chương trình Visual Basic. Các thuộc tính xác định cách nhìn và xử lý điều khiển. Những điều khiển giúp người dùng tương tác với chương trình. Khi thiết kế chương trình Visual Basic, bạn nên theo các chuẩn sau, khi có thể, để giảm tối đa công việc bảo trì sau này. Bạn đã thấy rằng:

- Ý kiến hay khi thiết kế trình ứng dụng là bạn không nên dùng nhiều điều khiển chỉ làm trình ứng dụng trông chật chội và phức tạp hơn mà thôi.
- Sử dụng 3 ký tự viết tắt tiêu chuẩn khi bạn đặt tên điều khiển để có thể xác định chính xác tên của điều khiển đang làm việc.
- Tuân theo chuẩn lập trình Windows để trình ứng dụng của bạn xử lý giống các ứng dụng Windows khác. Bằng cách đó, người dùng sẽ dễ làm quen, học tập và sử dụng các chương trình của bạn.

Bài thực hành cuối mỗi bài trình bày cách tạo chương trình Visual Basic. Chúng minh họa các khái niệm đã trình bày trong từng bài. Bắt đầu từ bài tiếp theo, bạn sẽ học chi tiết các điều khiển và thuộc tính. Trước đây, bạn chỉ làm quen cách thiết đặt và kích hoạt các điều khiển nhãn và nút lệnh. Có nhiều điều để học về các điều khiển. Bài thực hành này tập trung giới thiệu giúp bạn hiểu sâu sắc hơn và thực hành cách sử dụng Visual Basic.

Mô tả chương trình

Hình P2.1 mô tả màn hình đầu tiên của ứng dụng trong bài thực hành này. Các ứng dụng tỏ ra đơn giản và dễ thực hiện nếu bạn tuân thủ những chỉ dẫn kế tiếp.



Hình P2.1. *Sẵn sàng xem các thuộc tính khác.*

Mục đích chương trình này là minh họa các thuộc tính khác nhau có thể có trong nhãn. Bài 5 sẽ tìm hiểu chi tiết những thuộc tính của điều khiển nhãn, nhưng giờ đây bạn đã đủ khéo léo trong Visual Basic để hiểu các thuộc tính được minh họa trong chương trình này.

Hoạt động chương trình

Chương trình đầu tiên hiển thị một nhãn ở giữa màn hình. Viền nhãn cho biết nhãn khá rộng. Văn bản ở giữa nhãn thì nhỏ và nền nhãn có màu trắng. Nhấp nút lệnh Next Property để xem các thuộc tính khác nhau.

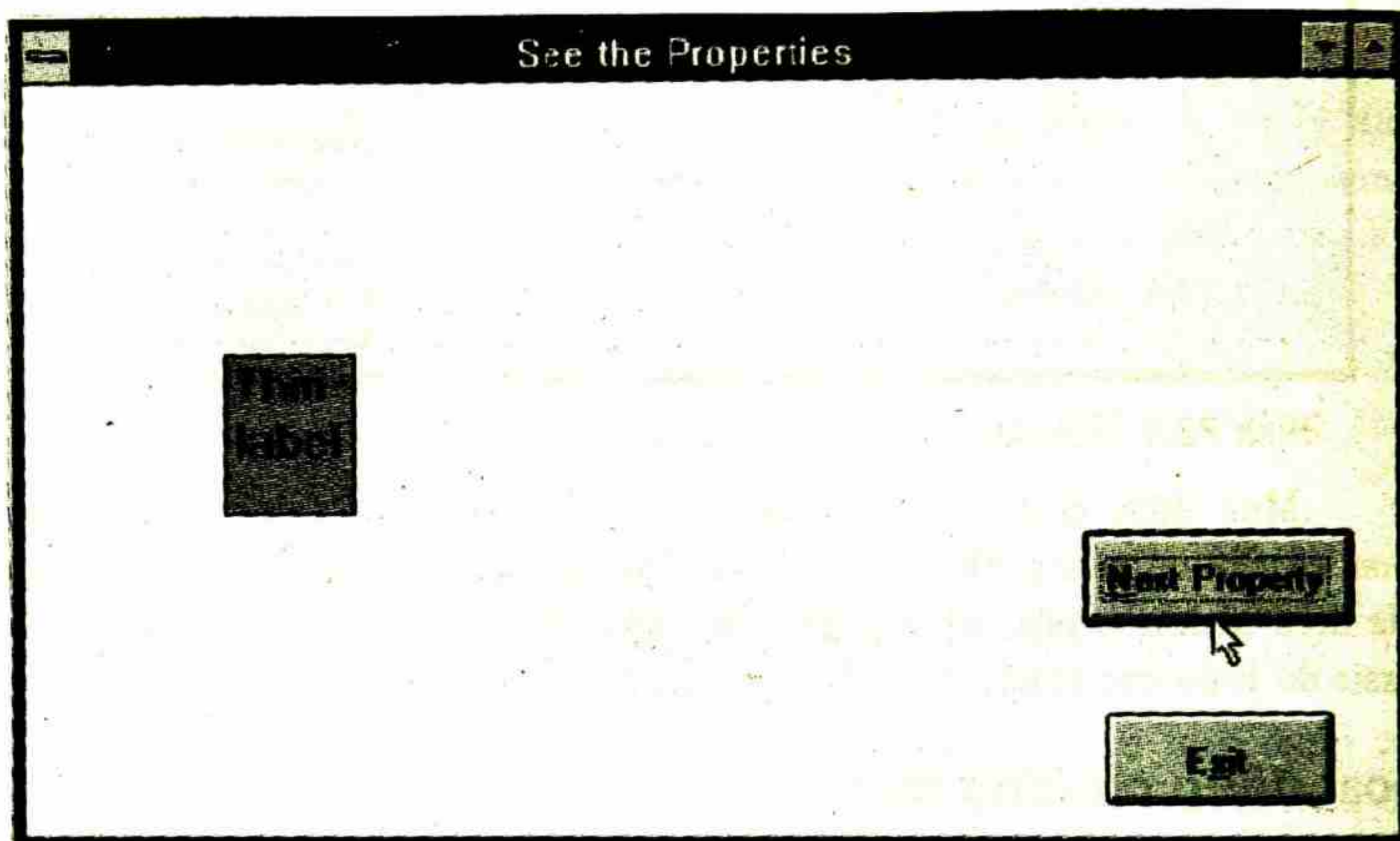
Thuộc tính thay đổi đầu tiên khi nhấp nút lệnh là Alignment. Nhãn có thể được canh trái, canh giữa hay canh phải bên trong độ rộng của nó. Nhãn bắt đầu được canh giữa, sau khi nhấp nút lệnh, bạn thấy nhãn canh trái.

Nhấp lại Next Property. Thuộc tính Alignment đổi thành canh phải.

Nhấp nút Next Property lần nữa, nhãn sẽ trở lại canh giữa và thay đổi thuộc tính kích cỡ phông chữ thành chữ lớn. Khi nhấp tiếp nút Next Property, bạn sẽ thấy nhãn được cập nhật để mô tả thuộc tính bạn đã thay đổi.

Để xem màu nền nhãn thay đổi, nhấp lại nút Next Property. Nền của nhãn đổi thành màu xanh lá cây sáng.

Nhấp nút lệnh Next Property một lần nữa để xem thay đổi thuộc tính nhãn cuối cùng. Như mô tả trong Hình P2.2, độ rộng của nhãn co lại thành kích cỡ nhỏ hơn. Khi bạn nhấp lại nút lệnh Next Property, nhãn sẽ trở về các giá trị thuộc tính kích cỡ, phông chữ và màu ban đầu.



Hình P2.2. *Thuộc tính độ rộng làm nhãn trở nên nhỏ hơn.*

Lưu ý

Ứng dụng trong bài thực hành này gồm một thủ tục biến cố sẽ thay đổi tất cả thuộc tính bất cứ khi nào người dùng nhấp nút lệnh Next Property. Bài thực hành không liệt kê mã lệnh cho thủ tục biến cố bởi vì mã lệnh hoàn toàn không có ý nghĩa gì vào lúc này. Nếu bạn thực sự muốn xem mã lệnh, hãy mở cửa sổ Code sau khi khởi động chương trình.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát chương trình bằng cách nhấp nút lệnh Exit. Ứng dụng sẽ thoát và bạn trở lại môi trường phát triển Visual Basic.

Chương III

Bài 5

Sử dụng nhãn, nút lệnh và hộp nhập

- **Tiêu điểm và điều khiển**
- **Nút lệnh**
- **Các thuộc tính nhãn**
- **Điều khiển hộp nhập**
- **Thiết đặt thuộc tính**

Bài này khảo sát tỉ mỉ nhãn, nút lệnh và hộp nhập đồng thời giải thích tất cả thuộc tính liên quan tới các điều khiển này. Điều quan trọng là bạn phải nhớ các thuộc tính chủ yếu khi đọc qua các bảng thuộc tính nếu bạn muốn làm chủ Visual Basic. Trừ khi bạn hiểu cách khởi tạo từng thuộc tính để đáp ứng những yêu cầu khi thiết kế các trình ứng dụng, còn không bạn sẽ không thể sử dụng hiệu quả Visual Basic, cũng như không thể tạo ra các ứng dụng đáng chú ý mà Visual Basic có khả năng thiết kế. Sau khi hiểu rõ các thuộc tính có hiệu lực với từng điều khiển, bài kế tiếp sẽ giải thích tất cả thủ tục biến cố có thể có với nút lệnh, nhãn và hộp nhập.

Lưu ý

Từ chương này đến cuối sách, mỗi bài thực hành đều bao gồm một ứng dụng minh họa, sử dụng các khái niệm đã nêu trong chương. Nội dung chương được xem như để trình bày phần lý thuyết, và bài thực hành là phần thực hành. Bạn sẽ sử dụng những gì đã học trong chương, đồng thời chỉ cho bạn ứng dụng thực tế. Bạn nên nắm cả hai phần lý thuyết và ứng dụng thực hành cuối chương.

TIÊU ĐIỂM ĐIỀU KHIỂN

Khái niệm

Khi học về điều khiển, bạn thường nghe thuật ngữ *tiêu điểm* (focus). Tìm hiểu về tiêu điểm bây giờ sẽ giúp bạn tránh được nhiều rắc rối về sau.

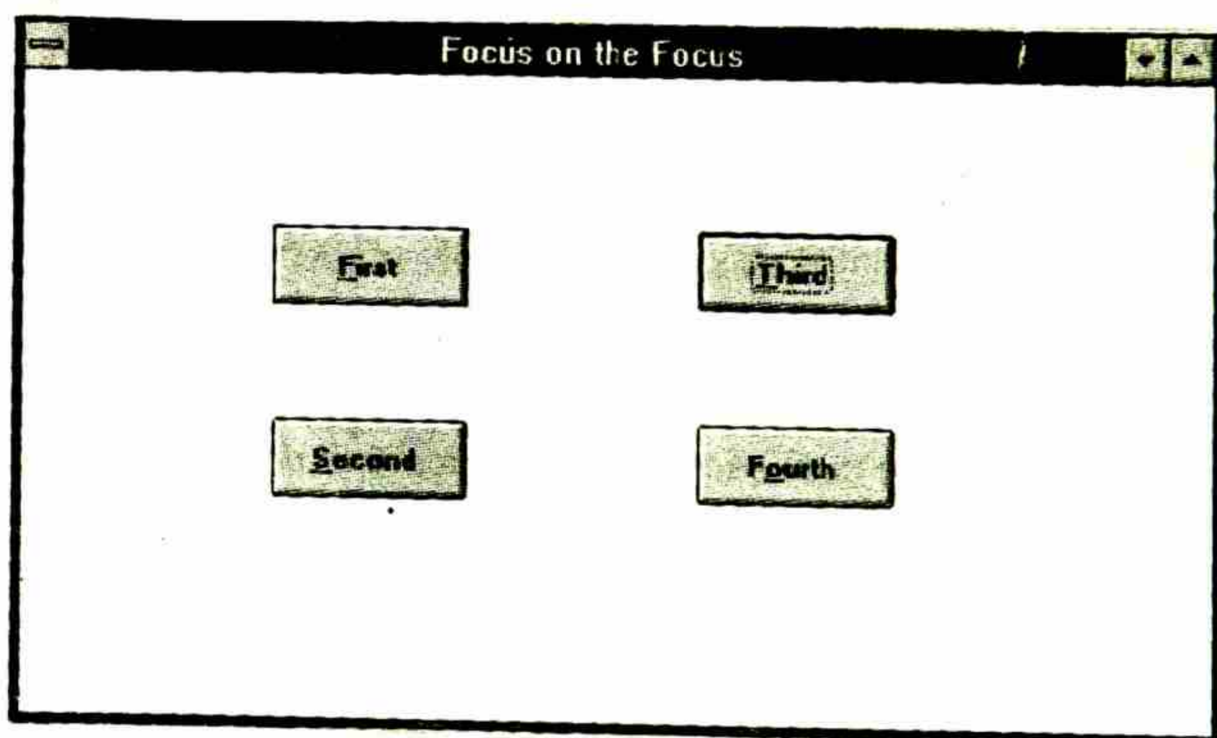
Khái niệm mới

Tiêu điểm (focus) nghĩa là chuyển quyền xử lý tới điều khiển đang được hiện sáng.

Điều khiển sẽ nhận tiêu điểm từ điều khiển kế cận và khi đó nó là điều khiển nhận đáp ứng của người dùng. Đa số người dùng Windows hiểu lờ mờ về tiêu điểm thậm chí ít khi nghĩ đến nó. Điều khiển có tiêu điểm luôn là điều khiển được hiện sáng. Tiêu điểm (focus) thường di chuyển từ điều khiển này tới điều khiển khác khi người dùng làm việc. Nó xác định nơi hành động kế tiếp sẽ thực hiện.

Mách nước

Theo cách này, một tiêu điểm của điều khiển giống như con trỏ văn bản của trình xử lý từ. Tiêu điểm (focus) cho phép người dùng biết nơi hành động kế tiếp sẽ thực hiện trừ khi có sự thay đổi tiêu điểm, cũng như con trỏ văn bản cho người dùng biết nơi ký tự kế tiếp sẽ xuất hiện trừ khi di chuyển con trỏ văn bản tới vị trí khác.



Hình 5.1. *Nút lệnh thứ ba có tiêu điểm.*

Tại thời điểm bất kỳ chỉ có một điều khiển duy nhất có tiêu điểm, và không phải điều khiển nào cũng có thể nhận tiêu điểm. Hình 5.1 chỉ ra mẫu biểu có bốn nút lệnh. Nút lệnh thứ ba có tiêu điểm (focus). Bạn có thể nói điều này bởi vì nút lệnh thứ ba được hiện sáng, như vậy những nút lệnh khác không có tiêu điểm. Không chỉ nút lệnh được hiện sáng mà còn có một đường chấm quanh đề mục của nó.

Khái niệm mới

Thứ tự tiêu điểm xác định điều khiển kế tiếp trong dãy sẽ nhận tiêu điểm.

Mỗi mẫu biểu có thứ tự tiêu điểm xác định điều khiển kế tiếp sẽ nhận tiêu điểm (focus). Trong Hình 5.1, nếu người dùng nhấn Enter, nút lệnh Third của hình sẽ chỉ lún xuống bởi vì nó có tiêu điểm. Nếu người dùng nhấn phím Tab trước khi nhấn Enter, điều khiển kế tiếp sẽ nhận tiêu điểm được hiện sáng. Nếu nút lệnh Second là điều khiển có thứ tự tiêu điểm kế tiếp trong dãy, việc nhấn phím Tab của người dùng sẽ hiện sáng nút lệnh Second.

Bạn đã thấy hộp thoại như hộp thoại File Open, nút OK bao giờ cũng là nút lệnh có tiêu điểm. Có thể nhấn Enter để chọn nút lệnh OK. Việc nhấp một nút lệnh khác (như nút lệnh Cancel chẳng hạn), sẽ giống việc nhấn Tab cho đến nút lệnh bạn muốn có tiêu điểm (focus).

Qua thiết đặt thuộc tính, bạn có thể xác định thứ tự tiêu điểm và quyết định liệu điều khiển có thể nhận tiêu điểm hay không. Mẫu biểu của bạn có thể sở hữu một số điều khiển mà bạn không muốn người dùng có khả năng hiện sáng, vì vậy, bạn phải ngăn không cho chúng nhận tiêu điểm.

Củng cố

Tiêu điểm (focus) xác định điều khiển kích hoạt kế tiếp. Nếu một nút lệnh có tiêu điểm, nó sẽ hiện sáng và được chọn nếu người dùng nhấn Enter. Nếu một hộp nhập có tiêu điểm, nó sẽ nhận ký tự nhấn phím kế tiếp ngay cả khi có năm hộp nhập khác trên mẫu biểu cùng thời điểm. Nếu người dùng cần nhập văn bản vào một hộp nhập xác định chưa nhận tiêu điểm, họ phải nhấn phím Tab cho đến khi hộp nhập đó nhận tiêu điểm, hoặc phải nhấp vào hộp nhập bằng con trỏ mouse.

NÚT LỆNH

Khái niệm

Nút lệnh (command button) là điều khiển không thể thiếu trong hầu hết trình ứng dụng Windows. Với nút lệnh, người dùng có thể trả lời các biến cố, phát tín hiệu khi sẵn sàng in cái gì đó và báo cho chương trình biết khi nào sẽ dừng. Tất cả nút lệnh hoạt động theo cùng một nguyên tắc – chúng trông như được ấn xuống khi người dùng nhấp mouse vào, và sẽ kích hoạt các biến cố, như biến cố Click chẳng hạn. Phần lớn thuộc tính của nút lệnh xác định duy nhất một hoạt động nào đó.

Bạn đã làm quen một số thuộc tính nút lệnh và đã thấy nút lệnh thay đổi kích cỡ và vị trí. Bảng 5.1 mô tả tất cả thuộc tính nút lệnh. Có lẽ bạn muốn mở một project mới. Hãy thêm nút lệnh vào cửa sổ Form bằng cách nhấp đúp biểu tượng nút lệnh trên hộp công cụ, và nhấn F4 để xem lướt qua các thuộc tính của nó trong cửa sổ Properties.

Khái niệm mới

Tên hằng là các giá trị được đặt tên.

Một vài xác lập thuộc tính có thể chấp nhận dãy giới hạn các giá trị số. Phần lớn các giá trị này được đặt tên hằng. Một số xác lập thuộc tính chấp nhận cả giá trị đúng (True) lẫn giá trị sai (False), để chỉ điều kiện yes hay no. Ví dụ, nếu thuộc tính Cancel thiết đặt là True, nút lệnh đó là nút lệnh Cancel và tất cả các nút lệnh khác phải có thuộc tính Cancel là False bởi vì chỉ có một nút lệnh trên mẫu biểu có thể có giá trị thuộc tính Cancel là True.

Mách nước

Khi đọc qua bảng thuộc tính trong sách, bạn không chỉ làm quen với mục đích của từng thuộc tính mà còn với tên của nó. Khi lập trình, bạn cần tham khảo các thuộc tính này với tên đầy đủ và chính xác.

Bảng 5.1. *Các thuộc tính nút lệnh*

Thuộc tính	Diễn giải
BackColor	Xác định màu nền của nút lệnh. Nhấp mũi tên xuống của bảng màu Backcolor để liệt kê danh sách các màu và nhấp System để xem danh sách màu Windows chung. Trước khi nút lệnh hiển thị màu nền, bạn phải thay đổi thuộc tính Style từ 0 – Standard thành 1 – Graphical.
Cancel	Nếu là True, Visual Basic tự động lấy biến cố Click của nút lệnh khi người dùng nhấn phím Esc. Tại một thời điểm, chỉ một nút lệnh có thuộc tính Cancel là True. Tất cả các nút lệnh còn lại có giá trị thuộc tính Cancel ban đầu là False.
Caption	Chữ sẽ xuất hiện trên nút lệnh. Nếu bạn đặt dấu & trước một ký tự, ký tự này trở thành phím truy xuất. Vì thế, phím truy xuất cho nút lệnh với thuộc tính Caption được ấn định là E&xit sẽ là Alt+X. Giá trị Caption mặc định là giá trị tên của nút lệnh.
Default	Nếu thuộc tính Default là True, nút lệnh sẽ đáp ứng biến cố. Nhấn phím Enter khi mẫu biểu được kích hoạt, dù điều khiển khác có tiêu điểm. Tất cả nút lệnh đều có giá trị thuộc tính Default ban đầu là False cho đến khi bạn thay đổi chúng.

Định nghĩa

***icon** là hình ảnh trên màn hình, hay còn gọi là biểu tượng.*

DragIcon	Icon sẽ xuất hiện khi người dùng kéo nút lệnh trên mẫu biểu

Bởi vì bạn hiếm khi cho người dùng di chuyển một nút lệnh, nên bạn sẽ không sử dụng nhiều xác lập thuộc tính Drag.

DragMode	Nếu giá trị là 1 – người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển – hay giá trị 0 (mặc định) cho việc kéo mouse tự động – người dùng không thể kéo nút lệnh, nhưng qua mã lệnh bạn có thể khởi tạo thao tác kéo nếu cần.
Enabled	Nếu là True (mặc định), nút lệnh có thể đáp ứng các biến cố. Ngược lại, Visual Basic dừng quá trình xử lý biến cố của riêng điều khiển đó.

FontBold	Nếu là True (mặc định), đề mục sẽ hiển thị các ký tự đậm.
FontItalic	Nếu là True (mặc định), đề mục sẽ hiển thị các ký tự nghiêng.
FontName	Tên kiểu phông chữ cho đề mục nút lệnh. Thông thường bạn sử dụng tên một loại phông chữ Windows.

Khái niệm mới

1 point *bằng 1/72 inch.*

FontSize	Kích thước tính theo point của phông chữ sử dụng trong đề mục nút lệnh.
FontStrikethru	Nếu là True (mặc định), đề mục hiển thị với các ký tự bị gạch bỏ. Nói cách khác, các ký tự có đường gạch ngang qua chúng.
FontUnderline	Nếu là True (mặc định), đề mục sẽ hiển thị các ký tự gạch dưới.
Height	Chiều cao của nút lệnh tính bằng twip.
HelpContextID	Nếu bạn thêm trợ giúp cảm ngữ cảnh vào trình ứng dụng, HelpContextID cung cấp số xác định cho văn bản trợ giúp.
Index	Nếu nút lệnh là một phần của mảng điều khiển, thuộc tính Index cung cấp giá trị chỉ số cho từng nút lệnh riêng biệt. (Xem Chương 6.)
Left	Khoảng cách tính bằng twip tính từ góc trái của cửa sổ Form tới góc trái của nút lệnh.
MousePointer	Hình dạng để con trỏ mouse thay đổi nếu người dùng di chuyển con trỏ mouse trên nút lệnh. Các giá trị có thể có từ 0 đến 12 và trình bày các hình dạng khác nhau mà con trỏ mouse có thể có. (Hãy xem Chương 12.)
Name	Tên của điều khiển. Mặc định, Visual Basic phát sinh các tên Command1, Command2, và v.v..., khi bạn thêm các nút lệnh tuần tự vào mẫu biểu.
TabIndex	Thứ tự tab tiêu điểm bắt đầu từ 0 và tăng lên mỗi lần bạn thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi các giá trị TabIndex của điều khiển. Không có 2 điều khiển trong cùng mẫu biểu có thể có cùng giá trị TabIndex.

TabStop	Nếu là True, người dùng có thể nhấn Tab để di chuyển tiêu điểm tới nút lệnh này. Nếu False, nút lệnh không thể nhận tiêu điểm.
Tag	Không được dùng với Visual Basic. Lập trình viên có thể dùng nó để xác định các chú thích được áp dụng cho nút lệnh.
Top	Khoảng cách twip từ cạnh trên cùng nút lệnh tới cạnh trên cùng mẫu biểu.
Visible	True hay False, cho biết liệu người dùng có thể thấy và sử dụng nút lệnh hay không.
Width	Độ rộng của nút lệnh tính theo twip.

Lưu ý

Bảng 5.1 chỉ liệt kê các thuộc tính nút lệnh mà bạn có thể khởi tạo và thay đổi trong cửa sổ Properties. Các thuộc tính nút lệnh khác mà bạn có thể thay đổi bằng việc sử dụng mã lệnh Visual Basic chỉ có tác dụng lúc chạy chương trình.

Củng cố

Có nhiều thuộc tính nút lệnh như bạn đã có dịp làm quen trong Bảng 5.1. Học các thuộc tính này sẽ giúp bạn dễ dàng nắm bắt những nút lệnh về sau.

CÁC THUỘC TÍNH NHÃN

Khái niệm

Nhãn giữ văn bản trên mẫu biểu. Mặc dù có một vài cách để hiển thị văn bản, song điều khiển nhãn (Label) cho phép bạn gửi các thông báo trên mẫu biểu vốn có thể thay đổi bằng mã lệnh Visual Basic. Tuy nhiên, người dùng không thể thay đổi giá trị của điều khiển nhãn.

Điều khiển nhãn là sự sống còn của những ứng dụng Visual Basic bởi vì bạn luôn luôn đặt văn bản trên mẫu biểu cho người dùng đọc. Sau đây là một số công dụng của điều khiển nhãn:

- Tạo các tiêu đề (hộp hay không hộp)
- Nội dung mô tả dữ liệu
- Thông báo cảnh báo về màu
- Nội dung mô tả đồ họa
- Các chỉ thị lệnh

Việc thiết đặt và khởi tạo nhãn vô cùng dễ dàng. Nếu bạn không muốn thông báo xuất hiện như một nhãn khi người dùng khởi động ứng dụng lần đầu tiên, thì nhớ xóa tất cả văn bản khỏi thuộc tính Caption của nhãn. Bảng 5.2 liệt kê các thuộc tính có hiệu lực đối với nhãn trong cửa sổ Properties.

Mách nước

Nhiều thuộc tính trong Bảng 5.2 tương tự trong Bảng 5.1. Một số thuộc tính như Width và Height, áp dụng cho nhiều điều khiển khác nhau. Các thuộc tính khác, như AutoSize, chỉ áp dụng cho một vài điều khiển.

Bảng 5.2. *Các thuộc tính nhãn*

Thuộc tính	Diễn giải
Alignment	Giá trị 0 – canh trái (mặc định), 1 – canh phải, hay 2 – canh giữa đề mục trong nhãn. Việc đặt một đường viền quanh nhãn hay tô màu nhãn không phụ thuộc giá trị thuộc tính canh lề này.
AutoSize	Nếu là True, điều khiển tự động điều chỉnh kích cỡ riêng của nó cho vừa với nội dung đề mục. Nếu là False (mặc định), điều khiển cắt bên phải văn bản nếu nhãn không đủ lớn để lưu toàn bộ đề mục.

Khái niệm mới

Số thập lục phân là một số dựa trên cơ số 16.

BackColor	Màu nền của nhãn. Đó là một số thập lục phân trình bày một trong hàng ngàn giá trị màu có thể có của Windows. Bạn dễ dàng chọn từ bảng màu được hiển thị bởi Visual Basic khi ấn định thuộc tính BackColor. Màu nền mặc định giống màu nền mặc định của mẫu biểu.
BackStyle	Nếu ấn định bằng 0, nghĩa là trong suốt, màu nền mẫu biểu có thể thấy qua nền nhãn. Nếu ấn định là 1 (mặc định), màu nền nhãn sẽ giấu mẫu biểu ra sau nhãn.
BorderStyle	Hoặc bằng 0 (mặc định) không có viền hoặc bằng 1 ứng với đường viền đơn cố định.
Caption	Chữ sẽ xuất hiện trên nhãn.
DragIcon	Biểu tượng sẽ hiển thị khi người dùng kéo điều khiển nhãn trên mẫu biểu.

Bởi vì bạn ít khi cho phép người dùng di chuyển một điều khiển nhãn, nên bạn sẽ không dùng các xác lập thuộc tính Drag... nhiều lắm.

DragMode	Bao gồm giá trị 1 ứng với các yêu cầu kéo mouse bằng tay – người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển – hoặc 0 (mặc định) để kéo mouse tự động – người dùng không thể kéo điều khiển nhãn, nhưng thông qua mã lệnh bạn có thể khởi tạo quá trình kéo nếu cần.
Enabled	Nếu là True (mặc định), điều khiển nhãn (Label) có thể đáp ứng biến cố. Ngược lại, Visual Basic sẽ dừng quá trình xử lý biến cố của điều khiển đó.
FontBold	Nếu là True (mặc định), đề mục sẽ hiển thị ở dạng chữ đậm.
FontItalic	Nếu là True (mặc định), đề mục sẽ hiển thị ở dạng chữ nghiêng.
FontName	Tên kiểu điều khiển nhãn. Thông thường bạn sử dụng tên phông chữ True Type trong Windows.
FontSize	Kích cỡ phông chữ tính theo point được dùng cho đề mục điều khiển nhãn.
FontStrikethru	Nếu là True (mặc định), đề mục sẽ hiển thị ở dạng chữ gạch bỏ. Nói cách khác, các ký tự có một đường kẻ ngang qua nó.
FontUnderline	Nếu là True (mặc định), đề mục sẽ hiển thị ở dạng có gạch dưới.
ForeColor	Màu văn bản trong đề mục.
Height	Chiều cao điều khiển nhãn tính bằng twip.
Index	Nếu điều khiển nhãn là thành phần của mảng điều khiển, thuộc tính Index cung cấp giá trị chỉ số cho từng điều khiển riêng biệt. (Xem Chương 6)
Left	Khoảng cách twip tính từ góc trái cửa sổ Form tới góc trái điều khiển nhãn.

Khái niệm mới

DDE ***viết tắt của*** **Dynamic Data Exchange.**

LinkItem	Bao gồm dữ liệu được chuyển tới trình ứng dụng DDE cao cấp.
LinkMode	Bằng 0 (mặc định) không cho phép DDE, bằng 1 cho phép DDE tự động, 2 cho phép DDE dựa trên mã lệnh, 3 cho yêu cầu thông báo dựa trên mã lệnh.
LinkTimeout	Đếm theo 1/10 giây thời gian đợi trả lời khi một thông báo DDE được gửi.
LinkTopic	Xác định trình ứng dụng nguồn và đề mục cho một trình ứng dụng DDE.
MousePointer	Hình dạng con trỏ mouse sẽ thay đổi nếu người dùng di chuyển con trỏ mouse trên điều khiển nhãn. Các giá trị có thể có từ 0 đến 12 và trình bày các hình dạng con trỏ mouse khác nhau (xem Chương 12).
Name	Tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên Label1, Label2, và ...., khi bạn lần lượt thêm điều khiển nhãn vào mẫu biểu.
TabIndex	Thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần bạn thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm (focus) bằng cách thay đổi giá trị TabIndex của điều khiển. Không có hai điều khiển trên cùng mẫu biểu có cùng giá trị TabIndex.
Tag	Không được dùng bởi Visual Basic. Lập trình viên có thể sử dụng nó để xác định các chú thích cho điều khiển nhãn.
Top	Khoảng cách twip từ cạnh trên cùng điều khiển nhãn tới cạnh trên cùng mẫu biểu.
Visible	True hay False, cho biết liệu người dùng có thể thấy và sử dụng điều khiển nhãn không.
Width	Độ rộng điều khiển nhãn theo đơn vị twip.
WordWrap	Nếu là True, văn bản cuộn để giữ nội dung đề mục. Nếu là False (mặc định), văn bản không cuộn mà bị cắt để vừa với đề mục.

Lưu ý

Bảng 5.2 chỉ liệt kê các thuộc tính nhãn mà bạn có thể khởi tạo và thay đổi trong cửa sổ Properties. Các thuộc tính nhãn khác mà bạn có thể thay đổi bằng cách dùng mã lệnh Visual Basic chỉ có hiệu lực lúc thi hành chương trình.

Hiếm khi, thậm chí không bao giờ, bạn phải xác định từng giá trị thuộc tính khi tạo mới một điều khiển. Đa số các giá trị mặc định làm việc tốt mà không cần bất kỳ sự điều chỉnh nào.

Củng cố

Các thuộc tính nhãn sẽ hiển thị văn bản trên mẫu biểu mà người dùng có thể đọc.

ĐIỀU KHIỂN HỘP NHẬP

Khái niệm

Điều khiển hộp nhập (Text Box) sẽ hiển thị các giá trị mặc định và nhận dữ liệu vào từ người dùng. Nó cho phép bạn xác định cách người dùng nhập vào dữ liệu và đáp ứng các câu hỏi và điều khiển mà bạn sẽ hiển thị.

Khi hiển thị hộp nhập trên mẫu biểu, bạn cho người dùng cơ hội nhận một giá trị mặc định – thuộc tính Text ban đầu của hộp nhập – hay thay đổi nó thành một điều gì khác. Người dùng có thể nhập vào văn bản thuộc bất kỳ loại dữ liệu nào – số, ký tự, và các ký tự đặc biệt. Người dùng có thể cuộn sang trái hay phải bằng cách dùng các phím mũi tên, và có thể dùng cả phím Ins và Del để chèn, xóa văn bản trong điều khiển hộp nhập.

Đa số thuộc tính hộp nhập làm việc giống các thuộc tính của điều khiển nhãn. Tuy nhiên, không giống nhãn, thuộc tính hộp nhập mô tả những thuộc tính nhập vào dữ liệu để điều khiển có thể làm việc với dữ liệu nhập vào của người dùng thay vì hiển thị văn bản đơn giản. Bảng 5.3 sẽ mô tả các giá trị thuộc tính cho điều khiển hộp nhập.

Bảng 5.3. *Các thuộc tính hộp văn bản*

Thuộc tính	Mô tả
Alignment	Ấn định là 0 để canh trái (mặc định), 1 để canh phải, hay 2 để canh giữa đề mục trong hộp nhập. Nếu MultiLine có giá trị False, Visual Basic sẽ bỏ qua xác lập Alignment.
BackColor	Màu nền hộp nhập. Đó là một số thập lục phân trình bày một trong số hàng ngàn giá trị màu Windows có thể có. Bạn có thể chọn từ bảng màu được hiển thị bởi Visual Basic ấn định thuộc tính BackColor. Màu nền mặc định giống màu nền mặc định mẫu biểu.

BorderStyle	Bằng 0 (mặc định) ứng với không có đường viền hoặc 1 ứng với đường viền đơn cố định.
DragIcon	Biểu tượng sẽ hiển thị khi người dùng kéo hộp nhập trên mẫu biểu.

Bởi vì bạn ít khi cho phép người dùng di chuyển hộp nhập, nên bạn không dùng xác lập thuộc tính Drag... nhiều lắm.

DragMode	Gồm giá trị 1 ứng với yêu cầu kéo mouse – người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển – hoặc bằng 0 (mặc định) cho quá trình kéo mouse tự động – người dùng không thể kéo hộp nhập, nhưng qua mã lệnh bạn có thể khởi tạo quá trình kéo nếu cần.
Enabled	Nếu là True (mặc định), hộp nhập có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng quá trình xử lý biến cố của điều khiển đó.
FontBold	Nếu là True (mặc định), thuộc tính Text sẽ hiển thị ở dạng chữ đậm.
FontItalic	Nếu là True (mặc định), thuộc tính Text sẽ hiển thị ở dạng chữ nghiêng.
FontName	Tên kiểu chữ của hộp nhập. Thông thường bạn dùng tên phông chữ True Type trong Windows.
FontSize	Kích cỡ phông chữ tính theo đơn vị point được dùng cho giá trị Text của điều khiển hộp nhập.
FontStrikethru	Nếu là True (mặc định), giá trị văn bản sẽ hiển thị ở dạng gạch ngang. Nói cách khác, các ký tự có đường gạch ngang qua chúng.
FontUnderline	Nếu là True (mặc định), giá trị văn bản sẽ hiển thị ở dạng gạch dưới.
ForeColor	Màu văn bản trong thuộc tính Text.
Height	Chiều cao của hộp nhập tính theo đơn vị twip.
HelpContextID	Nếu thêm trợ giúp cảm ngữ cảnh nâng cao vào trình ứng dụng của bạn, thuộc tính HelpContextID sẽ cung cấp giá trị số xác định văn bản trợ giúp.

HideSelection	Duy trì văn bản được hiện sáng ngay cả khi hộp nhập mất tiêu điểm của nó.
Index	Nếu hộp nhập là thành phần của mảng điều khiển, thuộc tính Index sẽ cung cấp giá trị chỉ số cho từng hộp nhập riêng biệt (xem Chương 6).
Left	Khoảng cách twip từ góc trái cửa sổ Form tới góc trái hộp nhập.
LinkItem	Bao gồm dữ liệu được chuyển vào một trình ứng dụng DDE nâng cao.
LinkMode	Ấn định là 0 (mặc định) ứng với không cho phép DDE, 1 ứng với cho phép DDE tự động, 2 ứng với mã lệnh dựa trên DDE, 3 ứng với mã lệnh dựa trên yêu cầu thông báo.
LinkTimeout	Đếm thời gian 1/10 giây đợi trả lời sau khi gửi thông báo DDE.
LinkTopic	Xác định trình ứng dụng nguồn và đề mục cho một trình ứng dụng DDE.
MaxLength	Nếu ấn định bằng 0 (mặc định), giới hạn giá trị Text có thể dài tới 32.000 ký tự. Ngược lại MaxLength xác định số ký tự người dùng có thể nhập vào hộp nhập.
MousePointer	Hình dạng con trỏ mouse sẽ thay đổi nếu người dùng di chuyển con trỏ mouse trên hộp nhập. Các giá trị có thể có từ 0 tới 12 và trình bày các hình dạng con trỏ mouse khác nhau. (Xem Chương 12.)

Khái niệm mới

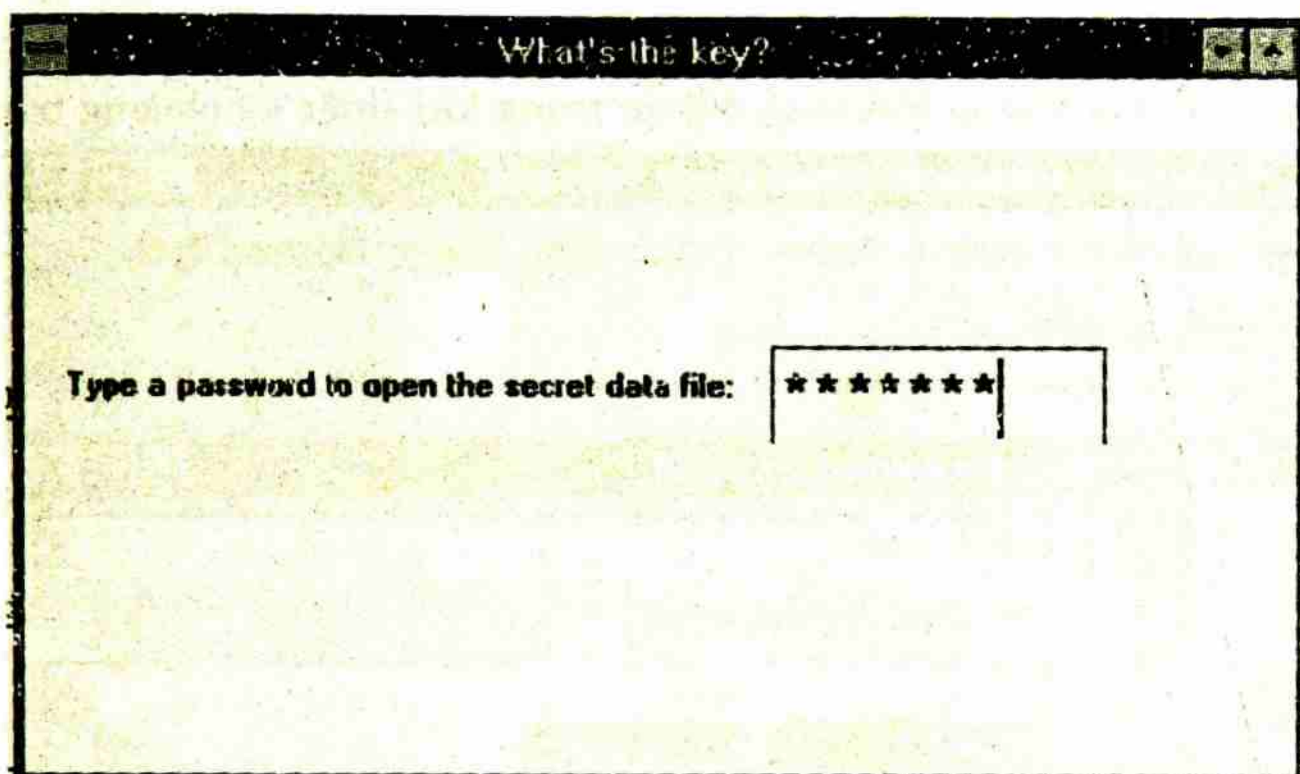
***Ký tự trở lại đầu dòng** sẽ gửi con trỏ văn bản tới dòng kế tiếp.*

MultiLine	Nếu là True, hộp nhập có thể hiển thị nhiều hơn một dòng văn bản. Nếu là False (mặc định), hộp nhập chỉ có một dòng văn bản. Văn bản có thể gồm cả ký tự trở lại đầu dòng.
Name	Tên điều khiển. Mặc định, Visual Basic sẽ phát sinh tên Text1, Text2, và ..., khi bạn lần lượt thêm các hộp nhập vào mẫu biểu.

PasswordChar	Nếu nhập vào một ký tự, ví dụ như dấu (*) cho PasswordChar, Visual Basic sẽ không hiển thị văn bản người dùng mà thay vào đó sẽ hiển thị PasswordChar khi người dùng nhập vào văn bản. Sử dụng hộp nhập với PasswordChar khi người dùng cần nhập vào mật mã và bạn không muốn người khác nhìn lén. Hình 5.2 mô tả một dạng nhập mật mã sử dụng ký tự * thay cho ký tự mật mã. Ngay cả khi hộp nhập nhận các ký tự được nhập thực sự của người dùng, màn hình cũng chỉ hiển thị ký tự PasswordChar.
ScrollBars	Ấn định bằng 0 (mặc định) ứng với không có thanh cuốn, 1 ứng với thanh cuốn ngang, 2 ứng với thanh cuốn dọc, hay 3 ứng với cả hai thanh cuốn.
TabIndex	Thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần thêm một điều khiển mới. Có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi giá trị TabIndex của điều khiển. Không có hai điều khiển nào trên cùng mẫu biểu có thể có cùng giá trị TabIndex.
TabStop	Nếu là True, người dùng có thể nhấn Tab để di chuyển tiêu điểm tới điều khiển nhãn. Nếu là False, điều khiển nhãn không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Lập trình viên có thể dùng nó để xác định chú thích cho hộp nhập
Text	Giá trị khởi tạo mà người dùng có thể thấy trong hộp nhập. Giá trị mặc định là tên điều khiển. Giá trị sẽ tiếp tục cập nhật khi người dùng nhập vào văn bản mới trong thời gian thi hành.
Top	Khoảng cách twip từ góc trên hộp nhập tới góc trên mẫu biểu.
Visible	True hay False, cho biết liệu người dùng có thể nhìn thấy.
Width	Độ rộng hộp nhập tính theo twip.

Lưu ý

Bảng 5.3 chỉ liệt kê những thuộc tính hộp nhập mà bạn có thể khởi tạo và thay đổi trong cửa sổ Properties. Những thuộc tính hộp nhập khác mà bạn có thể thay đổi bằng cách dùng mã lệnh Visual Basic chỉ có hiệu lực trong thời gian thi hành.



Hình 5.2. *Ký tự mật mã hiển thị thay cho ký tự người dùng nhập vào.*

Củng cố

Điều khiển hộp nhập cho phép bạn hiển thị các giá trị mặc định và chấp nhận những thay đổi, bổ sung mà người dùng thực hiện với các giá trị này khi nhập vào văn bản từ bàn phím. Một khi người dùng nhập vào văn bản trong một điều khiển hộp nhập (Text Box), bạn có thể kiểm tra thuộc tính Text để truy xuất văn bản đó.

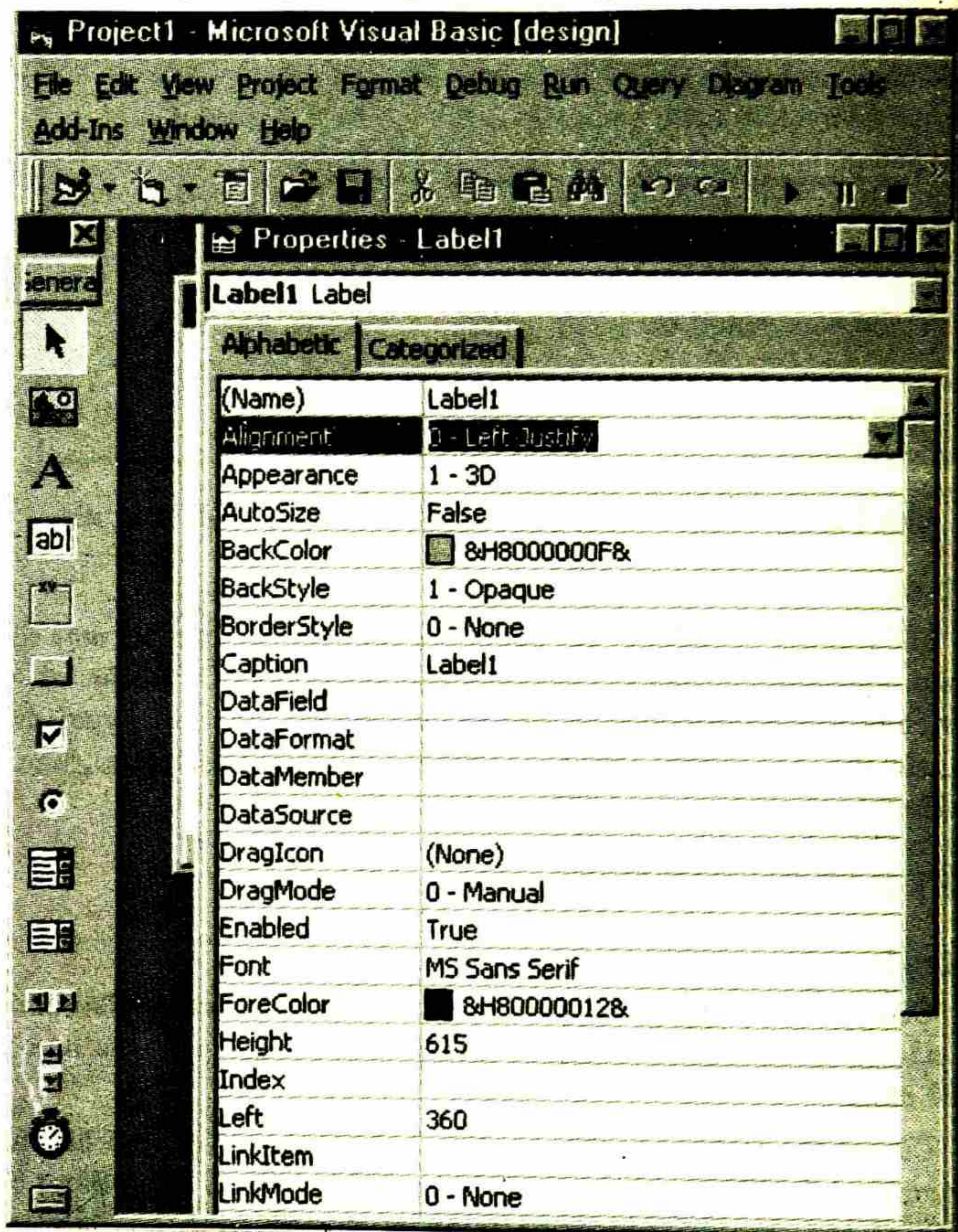
THIẾT ĐẶT THUỘC TÍNH

Khái niệm

Cửa sổ Properties giúp bạn ấn định thuộc tính cho các điều khiển trên mẫu biểu, và làm cho công việc thiết kế chương trình của bạn trở nên dễ dàng.

Cửa sổ Properties gồm danh sách thuộc tính có thể định lúc thiết kế chương trình. Chẳng hạn, có 25 thuộc tính nút lệnh (xem Bảng 5.1) mà bạn có thể thay đổi trong lúc thiết kế chương trình khi thiết đặt một nút lệnh lên mẫu biểu. Bạn cũng có thể đọc và thay đổi nhiều thuộc tính trong lúc chương trình vận hành. Ba thuộc tính có thể ấn định trong thời gian thi hành chương trình thông qua mã lệnh, sẽ

không có hiệu lực lúc thiết kế chương trình. Bạn có thể thay đổi các thuộc tính bổ sung này trong thời gian thi hành chương trình thông qua mã lệnh, nhưng không thể thay đổi nó trong khi thiết kế chương trình.

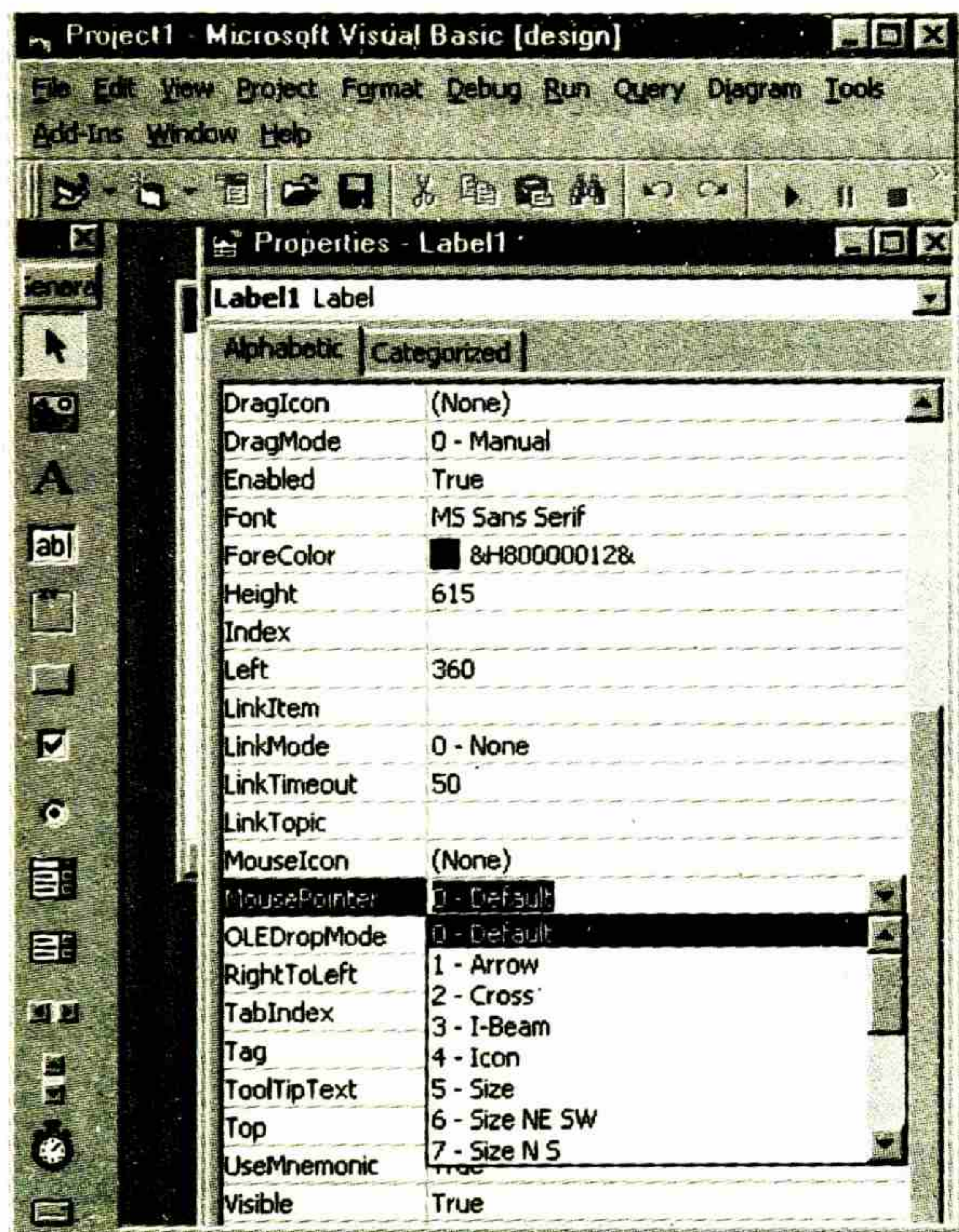


Hình 5.3. *Cửa sổ Properties cung cấp cho bạn nhiều cách thay đổi các giá trị thuộc tính.*

Cửa sổ Properties cung cấp nhiều cách thiết đặt giá trị thuộc tính. Hình 5.3 mô tả cửa sổ Properties của điều khiển nhãn. Lưu ý rằng cửa sổ Properties gồm có danh sách thuộc tính nhãn có thể cuộn lúc

thiết kế chương trình, vùng nhập liệu, danh sách xổ xuống giúp quá trình thiết đặt các giá trị thuộc tính, và danh sách xổ xuống dành cho quá trình chọn các thuộc tính của điều khiển khác.

Bạn có thể thay đổi một giá trị thuộc tính bằng cách nhấp hàng thuộc tính trong cửa sổ Properties và nhập vào một giá trị mới. Hộp nhập nội dung dữ liệu của cửa sổ sẽ nhận thuộc tính mới khi bạn nhập vào. Nếu bạn nhấp thuộc tính Name và nhập một tên mới, bạn sẽ thấy tên trong hộp nhập nội dung dữ liệu.

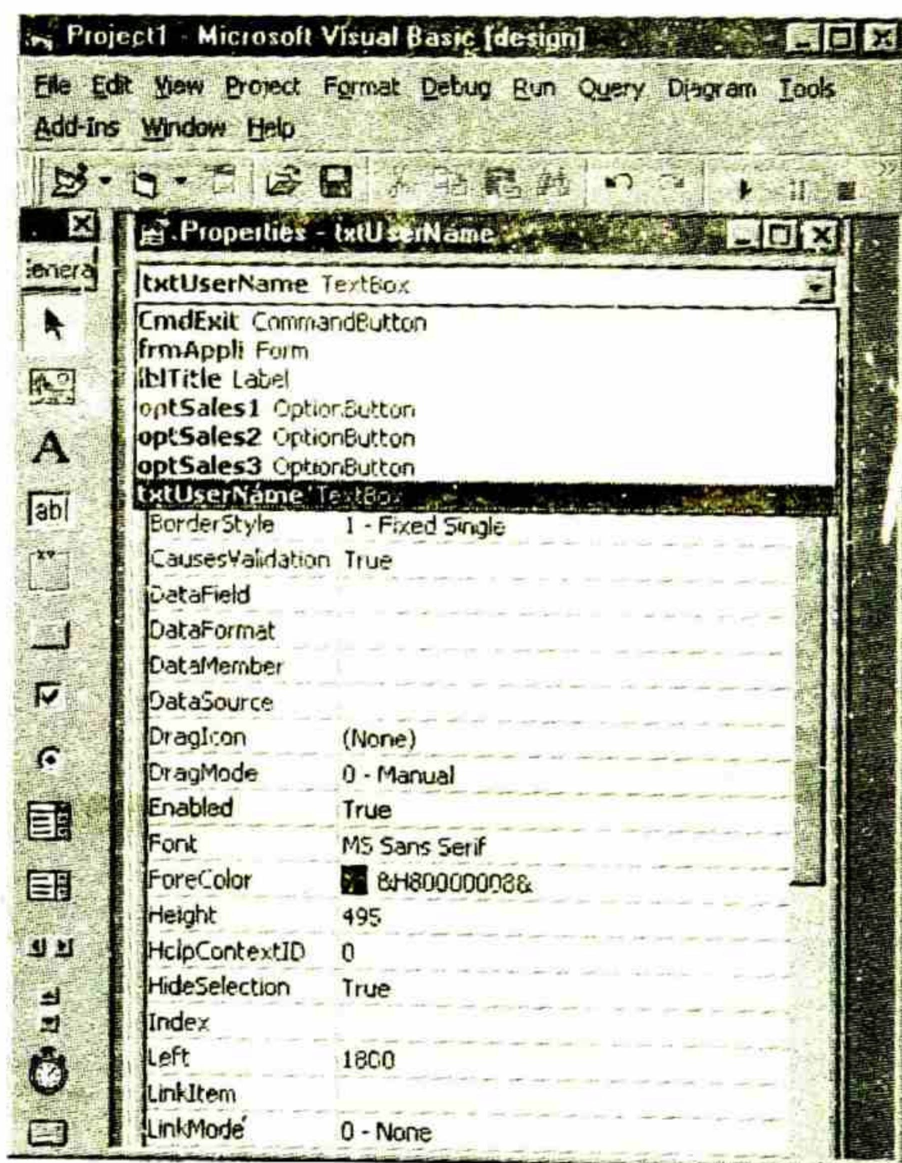


Hình 5.4. *Visual Basic đề nghị các giá trị thuộc tính để chọn.*

Cửa sổ Properties cũng cung cấp cho bạn nhiều giá trị để chọn nếu thuộc tính có một số giới hạn về giá trị. Chẳng hạn, thuộc tính MousePointer có thể giữ các giá trị từ 0–Default đến 7–Size NS, tương ứng với các hình dạng con trỏ mouse khác nhau có thể hiển thị. Thay vì nhập vào một trong bảy giá trị này, đơn giản bạn chỉ cần nhấp thuộc tính MousePointer và sau đó nhấp mũi tên xuống để mở danh sách xổ xuống của hộp nhập nội dung dữ liệu, như minh họa trong Hình 5.4. Bạn có thể chọn một giá trị từ danh sách mà không phải nhập giá trị nào.

Ghi chú

Để xem các thuộc tính điều khiển trong cửa sổ Properties, bạn không cần phải chọn điều khiển trước.



Hình 5.5. *Chọn điều khiển muốn thay đổi các thuộc tính của nó.*

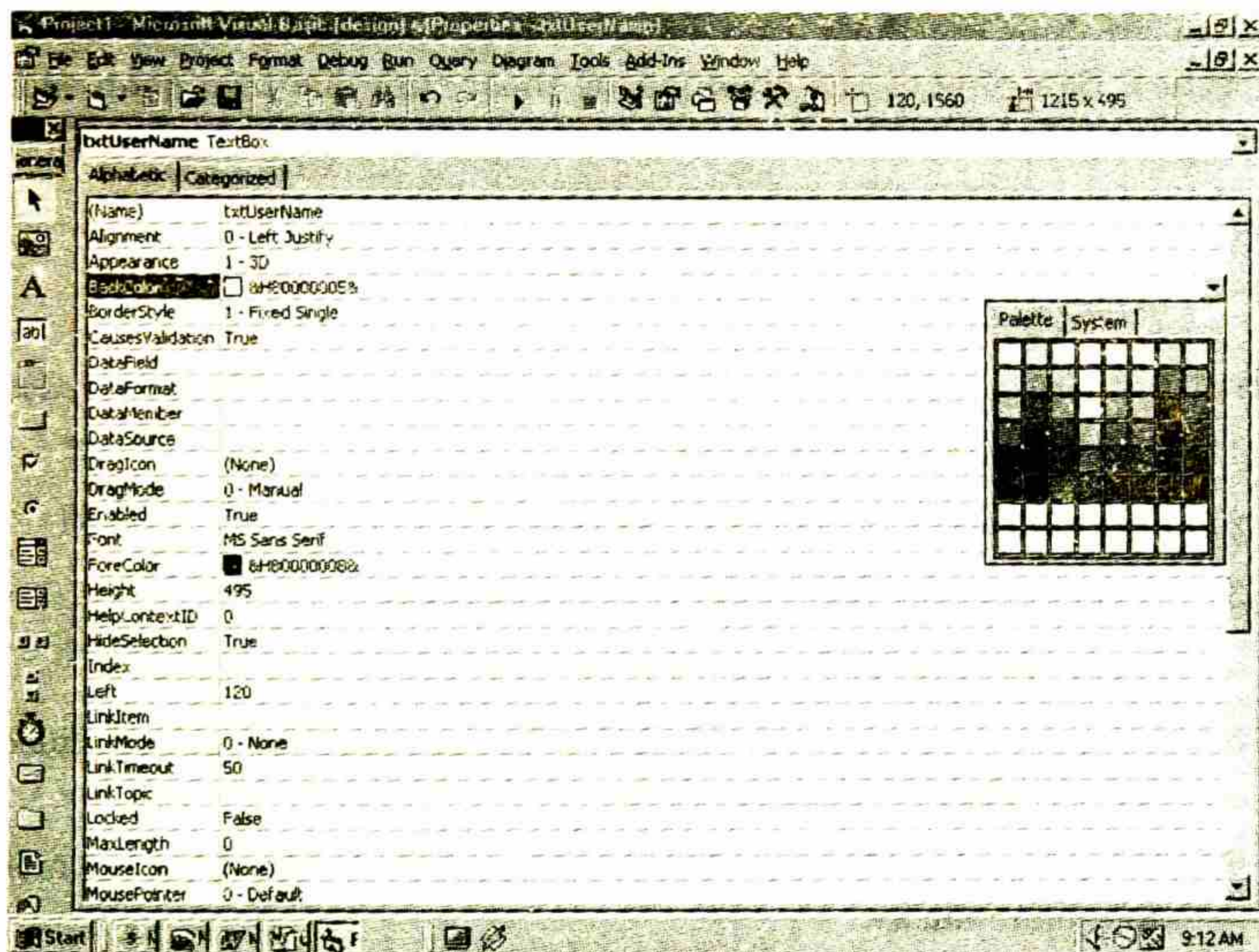
Một số lập trình viên thích thiết đặt nhiều điều khiển trên mẫu biểu trước khi ấn định các giá trị thuộc tính cho chúng. Sau khi bạn đặt nhiều điều khiển lên trên mẫu biểu, mỗi điều khiển có tập hợp thuộc tính riêng của nó mà bạn có thể xác định từ cửa sổ Properties. Hãy sử dụng hộp chọn điều khiển xổ xuống của Window để hiển thị mọi điều khiển trong trình ứng dụng, như được minh họa trong Hình 5.5. Chọn điều khiển từ danh sách các điều khiển xổ xuống và sẽ thấy những xác lập thuộc tính của nó hiển thị trong cửa sổ Properties.

Mách nước

Danh sách điều khiển xổ xuống trong cửa sổ Properties cho biết loại và tên của điều khiển.

Khái niệm mới

Bảng màu (palette) là tập hợp màu bạn có thể chọn.



Hình 5.6. *Chọn màu từ bảng màu.*

Khi thiết đặt thuộc tính màu, chẳng hạn như thuộc tính BackColor của nhãn, bạn có thể nhập một giá trị thập lục phân, nhưng cách dễ dàng hơn là chọn màu từ bảng màu. Visual Basic sẽ hiển thị cho bạn. Nếu đã nhấp thuộc tính BackColor của một điều khiển, chẳng hạn như điều khiển nhãn, hộp nhập nội dung dữ liệu của cửa sổ Properties có mũi tên ứng với hộp xổ xuống. Khi nhấp hình mũi tên, xong chọn mục Palette, Visual Basic sẽ hiển thị một bảng màu như Hình 5.6. Hãy nhấp một màu trong bảng để chọn màu cho thuộc tính BackColor. Visual Basic sẽ gán giá trị thập lục phân phù hợp với màu bạn chọn.

Củng cố

Cửa sổ Properties cung cấp nhiều loại trợ giúp nhanh cho bạn sử dụng khi thiết đặt các giá trị thuộc tính dành cho những điều khiển trên mẫu biểu. Bạn có thể di chuyển từ điều khiển này tới điều khiển khác và ấn định các thuộc tính bằng cách chọn điều khiển từ cửa sổ Properties. Bạn cũng có thể chọn giá trị thuộc tính dựa trên danh sách các giá trị cố định có thể chọn.

Bài tập

Kiến thức tổng quát

1. Tiêu điểm (focus) nghĩa là gì?
2. Bạn hiểu thế nào về thứ tự tiêu điểm?
3. Thuộc tính nào điều khiển thứ tự tiêu điểm?
4. Thuộc tính nào phổ biến và là tên của tất cả điều khiển?
5. Đúng hay Sai: Bạn có thể thiết đặt hay thay đổi tất cả thuộc tính lúc thiết kế chương trình.
6. Đúng hay Sai: Chỉ một điều khiển có thể có tiêu điểm tại một thời điểm bất kỳ.
7. Icon (biểu tượng) là gì?
8. Giá trị đơn vị *point*.

9. Bảng màu (palette) là gì?
10. Người dùng có thể nhập vào điều khiển hộp nhập những gì?
    A. Số
    B. Ký tự
    C. Ký tự đặc biệt
    D. Tất cả các kiểu dữ liệu trên
11. Thuộc tính nào làm cho điều khiển hộp nhập không thể kích hoạt thủ tục biến cố?
12. Vùng giá trị mà thuộc tính Alignment có thể lưu là gì?
13. Đúng hay Sai: Bạn phải đặt phần lớn các giá trị thuộc tính mỗi lần đặt một điều khiển lên mẫu biểu.
14. Ký tự trở về đầu dòng là gì?
15. *DDE* viết tắt của từ gì?
16. Đúng hay Sai: Một điều khiển có thể xuất hiện trên mẫu biểu nhưng người dùng không thấy được.
17. Bốn thuộc tính có hiệu lực cho phần lớn các điều khiển mô tả hai bộ chỉ dẫn đơn vị đo ở bên phải thanh công cụ chuẩn là gì?

Lập trình...

18. Điều khiển dùng để thiết kế các tiêu đề và chỉ thị tốt nhất là gì?
19. Xác lập thuộc tính nào của hộp nhập mà bạn sẽ sử dụng để nhận một thông báo bảo mật từ người dùng?
20. Giả sử bạn đang viết chương trình Visual Basic cần thi hành nhiệm vụ xác định khi người dùng nhấn phím Esc. Thuộc tính nào bạn phải thiết đặt?

Tìm lỗi kỹ thuật

21. An Huy không vui lắm. Anh ấy mới bắt đầu lập trình trong Visual Basic cho nên cần sự giúp đỡ để tháo gỡ những khó khăn gặp phải. An Huy đặt các điều khiển lên mẫu biểu trước khi thiết đặt những giá trị thuộc tính cho điều khiển. Vấn đề duy nhất mà An Huy gặp phải là anh ấy nghĩ rằng phải đóng cửa sổ Properties, hiện sáng điều khiển khác, rồi hiển thị lại cửa sổ Properties mới thiết đặt các thuộc tính cho điều khiển đó. Hãy chỉ cho anh ấy một cách tốt hơn.

Phần nâng cao

Mặc dù chỉ một điều khiển có thể nhận tiêu điểm (focus) tại một thời điểm, có một thuộc tính làm người dùng có thể nghĩ rằng hộp nhập vẫn nhận tiêu điểm khi tiêu điểm thực sự đang ở điều khiển khác. Tên thuộc tính này trong hộp nhập là gì? Thuộc tính nào của điều khiển hộp nhập làm việc giống thuộc tính Caption của một nút lệnh hay nhãn?

Bài 6

Đánh bóng mẫu biểu và điều khiển

- **Thuộc tính mẫu biểu**
- **Nhãn nâng cao**
- **Quá trình cuộn hộp nhập**
- **Sử dụng tiêu điểm để kiểm soát hộp nhập**
- **Biến cố điều khiển**

Bài này tiếp tục công việc của Bài 5, giải thích chi tiết hơn về mẫu biểu và các xác lập thuộc tính. Sau đó sẽ nghiên cứu sâu hơn về điều khiển nhãn và hộp nhập bằng cách mô tả cho bạn một số tiện ích nâng cao của các điều khiển này. Bạn không chỉ tìm hiểu về các xác lập thuộc tính, mà còn làm quen với những biến cố nào có thể sử dụng cho các điều khiển này.

Sau khi hoàn thành bài này, bạn sẽ biết hầu như mọi thứ về mẫu biểu, nút lệnh, nhãn, và hộp nhập. Bài thực hành cuối chương sẽ minh họa hành động của những điều khiển này.

THUỘC TÍNH MẪU BIỂU

Khái niệm

Mẫu biểu là đối tượng Visual Basic khác. Vì vậy, bạn có thể thiết đặt các thuộc tính cho nó trong khi thiết kế trình ứng dụng hay trong thời gian thi hành chương trình. Phần này sẽ giải thích chi tiết tất cả xác lập thuộc tính mẫu biểu.

Bảng 6.1 sẽ mô tả những xác lập thuộc tính mẫu biểu sẽ xuất hiện trong cửa sổ Properties khi bạn nhấp cửa sổ Form và nhấn F4. Mẫu biểu có nhiều thuộc tính hơn các điều khiển nút lệnh, nhãn và hộp nhập, các thuộc tính của nó bạn đã gặp ở bài trước. Giống các giá trị thuộc tính điều khiển, bạn đừng bao giờ lo lắng về tất cả thuộc tính này. Giá trị mặc định bao giờ cũng thỏa mãn trình ứng dụng của bạn.

Bảng 6.1. *Thuộc tính mẫu biểu*

Thuộc tính	Mô tả
AutoRedraw	Nếu là True, Visual Basic tự động vẽ lại các hình ảnh đồ họa trên mẫu biểu khi cửa sổ khác che mất hình ảnh hay khi người dùng định lại kích cỡ đối tượng. Nếu là False (mặc định), Visual Basic sẽ không tự vẽ lại.
BackColor	Màu nền mẫu biểu. Bạn có thể nhập vào giá trị màu hệ thập lục phân trong Windows hay chọn từ bảng màu.
BorderStyle	Ấn định là 0 nếu không có đường viền, 1 có đường viền đơn, 2 (mặc định) có đường viền khá lớn, 3 thiết đặt đường viền đôi cố định.
Caption	Văn bản hiển thị trong thanh tiêu đề mẫu biểu. Mặc định Caption là tên mẫu biểu.
ClipControls	Nếu là True (mặc định), biến cố Paint – biến cố vẽ lại được kích hoạt khi các hình ảnh đồ họa bị che sau đó hiện lại – vẽ lại các hình. Nếu là False, chỉ vùng mới hiển thị lại của hình được vẽ lại.
ControlBox	Nếu là True (mặc định), mẫu biểu có nút điều khiển và menu điều khiển. Nếu là False, mẫu biểu không có nút điều khiển và menu điều khiển.
DrawMode	Gồm 16 xác lập nâng cao liên quan tới các thuộc tính đang vẽ để xuất ra những hiệu ứng vẽ đặc biệt. (Xem Chương 11 để biết thêm thông tin về đồ họa.)
DrawStyle	Gồm 7 xác lập nâng cao xác định dáng vẻ đường kẻ.

Khái niệm mới

***Pixel** là độ rộng màn hình nhỏ nhất trên màn hình của bạn.*

DrawWidth	Độ rộng đường được vẽ trên mẫu biểu tính theo pixel.
Enabled	Nếu là True (mặc định), mẫu biểu có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng quá trình xử lý biến cố trên mẫu biểu.
FillColor	Giá trị màu dùng để tô các hình được vẽ trên mẫu biểu.
FillStyle	Gồm 8 kiểu xác định dáng vẻ các mẫu bên trong hình được vẽ trên mẫu biểu.

FontBold	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
FontItalic	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
FontName	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
FontSize	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
FontStrikethru	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
FontTransparent	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
FontUnderline	Không có hiệu lực với thuộc tính Caption của mẫu biểu, nhưng có hiệu lực với văn bản hiển thị trên mẫu biểu nếu bạn sử dụng lệnh Print.
ForeColor	Màu văn bản sẽ hiển thị trên mẫu biểu nếu sử dụng lệnh Print.
Height	Chiều cao của mẫu biểu tính theo twip.
HelpContextID	Cung cấp số xác định văn bản trợ giúp nếu bạn bổ sung đặc tính trợ giúp cảm ngữ cảnh nâng cao vào trình ứng dụng của bạn.
Icon	Biểu tượng người dùng nhìn thấy sau khi thu nhỏ tối đa mẫu biểu.
KeyPreview	Nếu là False (mặc định), điều khiển có tiêu điểm sẽ nhận các biến cố: KeyDown, KeyUp, và KeyPress trước khi mẫu biểu nhận biến cố. Nếu là True, mẫu biểu sẽ nhận các biến cố trước khi điều khiển nhận tiêu điểm.
Left	Khoảng cách tính theo twip từ góc trái màn hình tới góc trái mẫu biểu.

LinkMode	Ấn định là 0 (mặc định) không cho phép DDE, 1 cho phép DDE tự động, 2 cho phép DDE dựa trên mã lệnh, 3 yêu cầu khai báo dựa trên mã lệnh.
LinkTopic	Xác định trình ứng dụng nguồn và đề mục cho trình ứng dụng DDE.
MaxButton	Nếu là True (mặc định), nút Maximize sẽ hiển thị trên mẫu biểu trong lúc thi hành. Nếu là False, người dùng không thể phóng to cửa sổ Form.

Khái niệm mới

MDI ***viết tắt của giao diện nhiều tài liệu.***

MDIChild	Nếu là True, mẫu biểu là một mẫu biểu MDI – nghĩa là có mẫu biểu con trong mẫu biểu cha. Nếu là False (mặc định), mẫu biểu không phải là mẫu biểu MDI.
MinButton	Nếu là True (mặc định), nút Minimize sẽ hiển thị trên mẫu biểu trong thời gian chạy chương trình. Nếu là False, người dùng không thể thu nhỏ cửa sổ Form.
MousePointer	Hình dạng con trỏ mouse sẽ thay đổi nếu người dùng di chuyển nó trên mẫu biểu. Các giá trị có thể có từ 0 đến 12 và trình bày các hình dạng con trỏ mouse khác nhau. (Xem Chương 12.)
Name	Tên mẫu biểu. Mặc định, Visual Basic sẽ phát sinh tên Form1.
Picture	Tập tin hình hiển thị trên nền mẫu biểu.
ScaleHeight	Chiều cao mẫu biểu. ScaleMode xác định đơn vị đo được sử dụng.
ScaleLeft	Khoảng cách từ bên trái màn hình tới góc trái mẫu biểu. ScaleMode sẽ xác định đơn vị đo được dùng.
ScaleMode	Cho phép bạn xác định cách đo tọa độ trên mẫu biểu. Bạn có thể chọn một trong 8 giá trị. Đơn vị đo mặc định là twip, tương ứng giá trị 1. Đơn vị đo cho các thuộc tính Scale... khác sẽ là twip. Bảng 6.2 mô tả các đơn vị đo có thể có.
ScaleTop	Khoảng cách từ đỉnh màn hình tới góc trên mẫu biểu. ScaleMode xác định đơn vị đo sẽ sử dụng.
ScaleWidth	Độ rộng mẫu biểu. ScaleMode xác định đơn vị đo được sử dụng.

Tag	Không được dùng với Visual Basic. Lập trình viên có thể dùng nó để xác định các chú thích về mẫu biểu.
Top	Khoảng cách twip từ biên trên màn hình tới biên trên mẫu biểu.
Visible	True hay False, cho biết liệu người dùng có thể thấy và sử dụng mẫu biểu.
Width	Độ rộng mẫu biểu theo đơn vị twip.
WindowState	Mô tả tình trạng khởi động mẫu biểu khi người dùng chạy chương trình. Nếu ấn định là 0 (mặc định), ban đầu mẫu biểu sẽ hiển thị cùng kích cỡ đã thiết kế. Nếu là 1, mẫu biểu sẽ hiển thị dưới một biểu tượng thu nhỏ. Nếu là 2, mẫu biểu sẽ phóng to toàn màn hình.

ScaleMode cho phép bạn xác định cách đo các tọa độ trên mẫu biểu. Bạn có thể chọn 8 giá trị. Đơn vị đo mặc định là twip. Bảng 6.2 sẽ mô tả các đơn vị đo có thể có.

Bảng 6.2. *Các giá trị thuộc tính ScaleMode*

Giá trị	Mô tả
0	Các giá trị chuyên biệt
1	Twip (mặc định)
2	Point
3	Pixel
4	Ký tự chuẩn rộng 120 twip và cao 240 twip
5	Inch
6	Mm
7	Cm

Củng cố

Bạn có thể chuyên biệt mẫu biểu của mình theo nhiều cách, như phóng to toàn màn hình hay thu nhỏ thành một biểu tượng. Bạn dễ dàng sử dụng màu và các kiểu chữ khác nhau. Khi muốn một mẫu biểu đơn giản với đề mục xác định trình ứng dụng, giá trị thuộc tính duy nhất mà bạn sẽ phải điều chỉnh là Caption.

NHÃN NÂNG CAO

Khái niệm

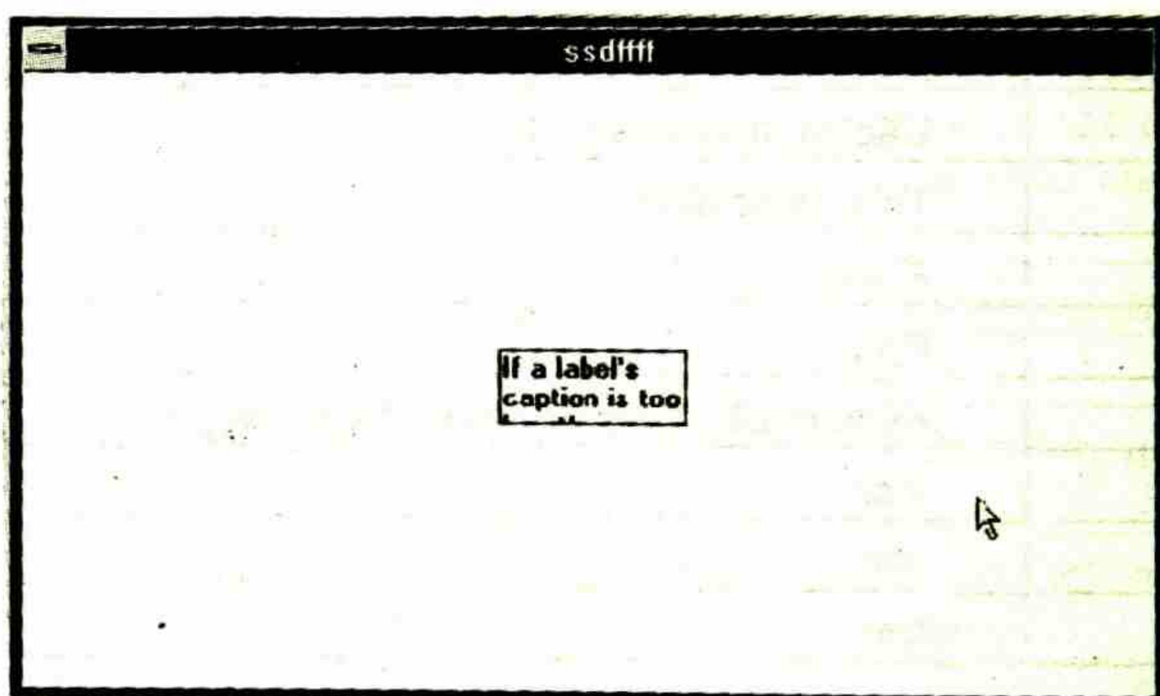
Trong bài trước, bạn đã biết rằng tất cả giá trị thuộc tính có thể thiết đặt với nhãn. Một số giá trị thuộc tính đem lại các hiệu ứng đáng quan tâm, sẽ được trình bày trong phần này. Giả sử bạn thiết kế nhãn bao gồm đề mục dài như sau:

> If a label's caption is too lengthy, you will need to adjust the label some way.

Một nhãn hiếm khi đủ rộng và cao để chứa đề mục này. Nếu bạn cố nhập văn bản vào trong thuộc tính Caption của nhãn nhưng lại dài hơn kích cỡ nhãn, những điều sau có thể xảy ra phụ thuộc vào cách khởi tạo nhãn của bạn:

Khái niệm mới

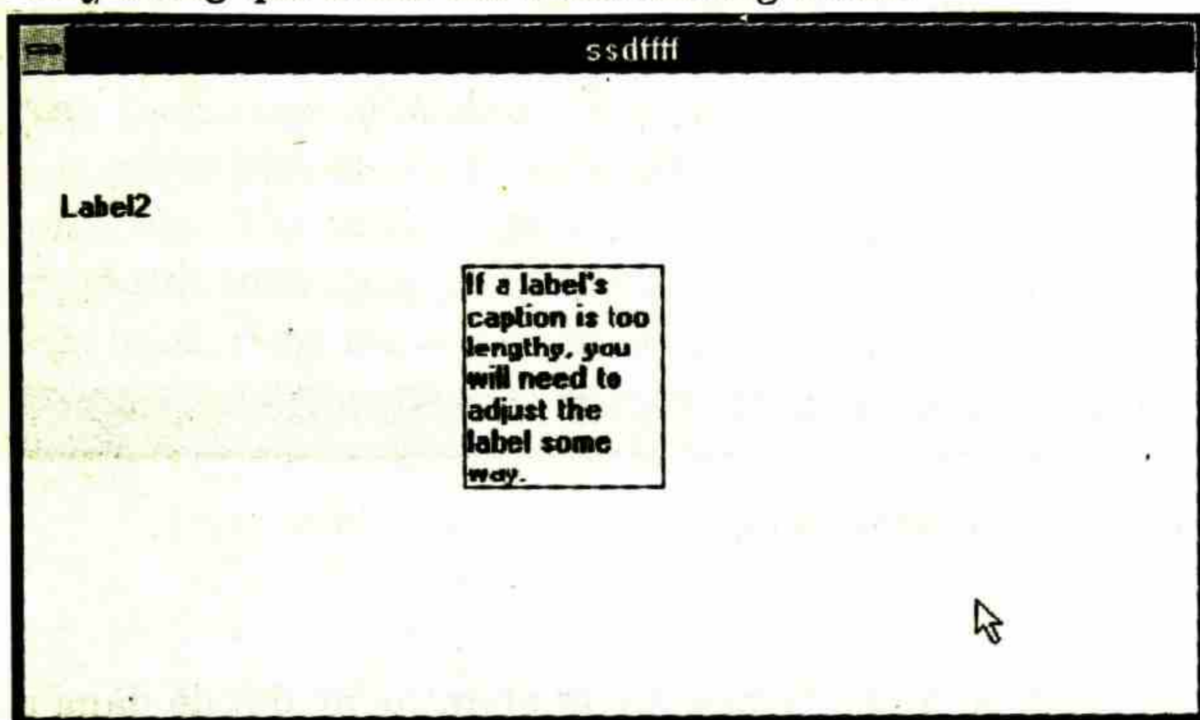
Truncate nghĩa là cắt xén bớt.



Hình 6.1. *Nhãn không đủ lớn.*

1. Nội dung không vừa bên trong nhãn, và Visual Basic sẽ cắt bớt. Hình 6.1 mô tả kết quả này. Thiết đặt thuộc tính AutoSize bằng False nếu muốn nhãn giữ nguyên kích cỡ. Trình ứng dụng sẽ gán văn bản, và nhãn có thể không giữ toàn bộ đề mục.

2. Nhãn tự động mở rộng xuống dưới để lưu toàn bộ đề mục trong một hộp nhiều dòng. Hình 6.2 mô tả kết quả này. Đặt thuộc tính AutoSize và WordWrap bằng True nếu bạn muốn nhãn mở rộng theo chiều dọc để lưu toàn bộ đề mục bạn gán lúc thiết kế hay trong quá trình thi hành chương trình.



Hình 6.2. *Nhãn định lại kích cỡ theo chiều dọc.*

Mách nước

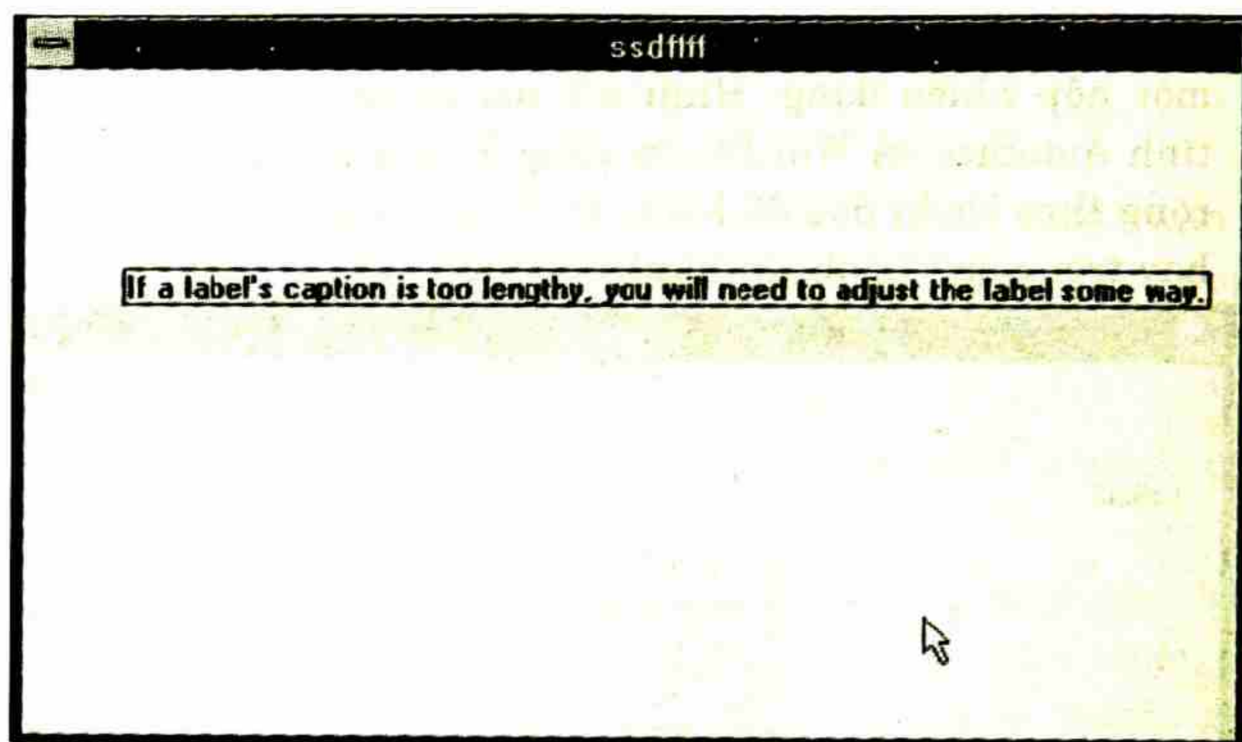
Thiết đặt thuộc tính WordWrap bằng True trước khi bạn ấn định thuộc tính AutoSize bằng True. Nếu bạn thiết đặt thuộc tính AutoSize trước, nhãn sẽ mở rộng theo chiều ngang trước khi bạn có cơ hội ấn định thuộc tính WordWrap.

Lưu ý

Hãy cẩn thận khi thiết lập quá nhiều nhãn tự điều chỉnh kích cỡ. Các nhãn có thể chồng lấp thông tin quan trọng trên mẫu biểu nếu đề mục của nó quá dài.

3. Nhãn tự động mở rộng theo chiều ngang để hiển thị toàn bộ đề mục trong một điều khiển nhãn dài. Hình 6.3 sẽ mô tả kết quả.

   Nhãn dài như vậy là không cần thiết. Tùy thuộc vào chiều dài văn bản bạn gán cho nhãn trong thời gian thi hành chương trình, có thể có nhiều khoảng trống để hiển thị các nhãn dài trên màn hình. Muốn tự động mở rộng nhãn theo chiều ngang, định thuộc tính AutoSize bằng True và thuộc tính WordWrap bằng False. Đây là giá trị mặc định.



Hình 6.3. *Nhãn không đủ cao.*

Củng cố

Quá trình định các đề mục trong nhãn tương đối dễ dàng cho đến khi bạn nhớ đến các hiệu ứng có thể xảy ra nếu nhãn quá lớn hay quá nhỏ để lưu nội dung văn bản. Bằng cách dùng các tổ hợp thuộc tính được mô tả, bạn có thể bổ sung các nhãn tự điều chỉnh kích cỡ cho phù hợp với nhãn cần lưu.

QUÁ TRÌNH CUỘN HỘP NHẬP

Khái niệm

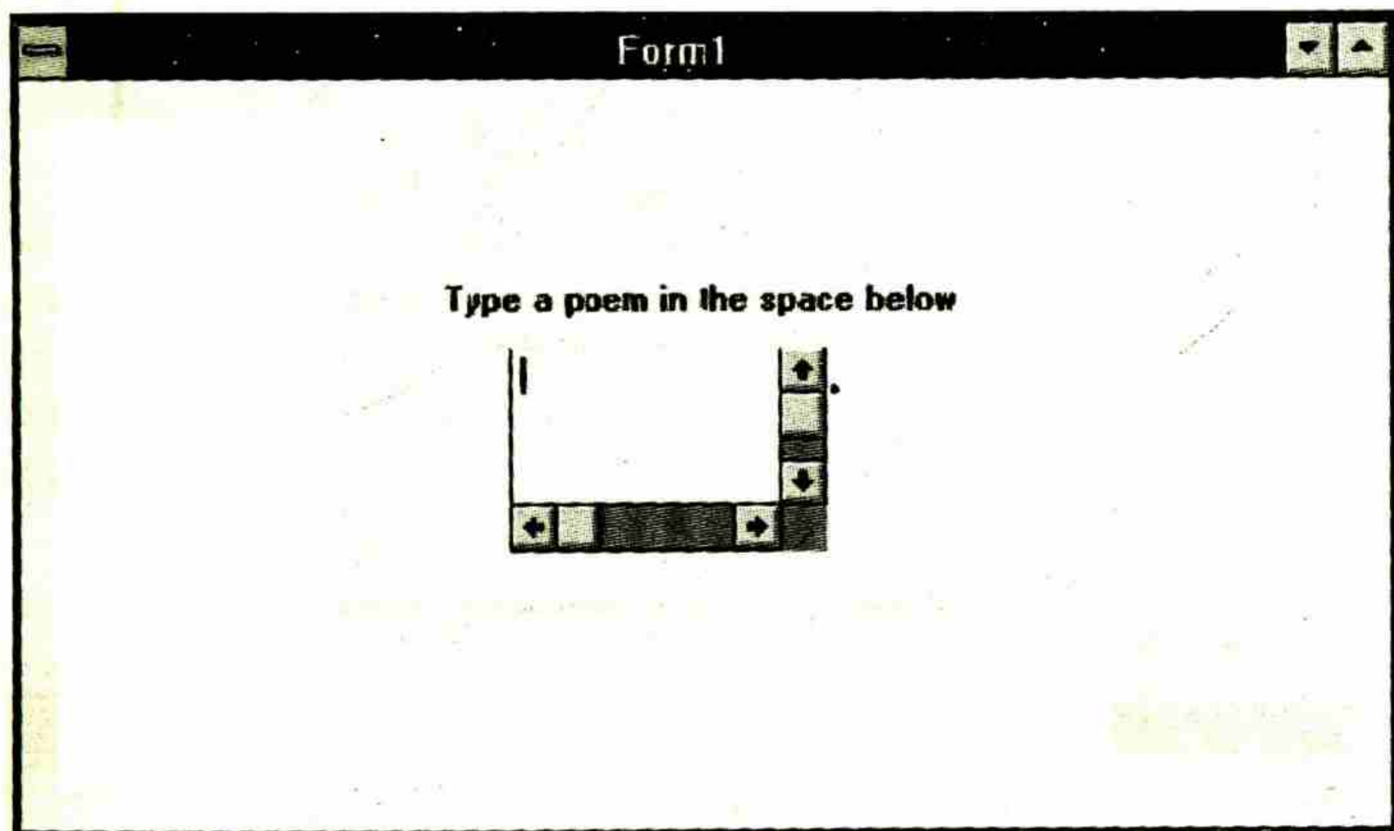
Bằng cách bổ sung các thanh cuốn vào hộp nhập, bạn có thể cung cấp cho người dùng hộp nhập nhiều dòng. Bằng cách đó, người dùng có thể nhập vào và điều chỉnh văn bản dài mà văn bản không bị che khuất trong hộp nhập.

Thuộc tính MultiLine của điều khiển hộp nhập sẽ xác định liệu hộp nhập có thể có một hay nhiều dòng văn bản hay không. Văn bản nhiều dòng có thể là giá trị mặc định ban đầu lưu trong thuộc tính Text của hộp nhập khi đặt mẫu biểu trên điều khiển. Văn bản nhiều dòng cũng có thể bao gồm dữ liệu nhập vào từ người dùng khi thi hành chương trình.

Lưu ý

Nếu định thuộc tính MultiLine bằng True, bạn cũng phải định thuộc tính Scrollbars với một giá trị khác 0–None. Người dùng phải có cách xem nhiều dòng văn bản trong hộp nhập và các thanh cuốn sẽ cung cấp cho người dùng khả năng đó.

Hình 6.4 mô tả một hộp nhập bao gồm các thanh cuốn và một giá trị True cho thuộc tính MultiLine. Khi người dùng nhập vào văn bản trong một hộp như vậy, họ có thể nhấn phím Enter để di chuyển tới dòng kế tiếp trong hộp. Tuy nhiên, người dùng không phải nhấn phím Enter bởi vì văn bản sẽ cuộn sang phải; thanh cuốn ngang cho phép cuộn trái và phải. Khi người dùng muốn tới cuối dòng trong hộp nhập, nhấn Enter.



Hình 6.4. *Các thanh cuốn của nhãn giúp người dùng thuận tiện hơn.*

Ghi chú

Bạn không thể nhập một giá trị khởi tạo cho hộp nhập nhiều dòng, mà chỉ có thể khởi tạo một hộp nhập với văn bản chiếm nhiều dòng trong hộp nhập trong thời gian thi hành chương trình.

Củng cố

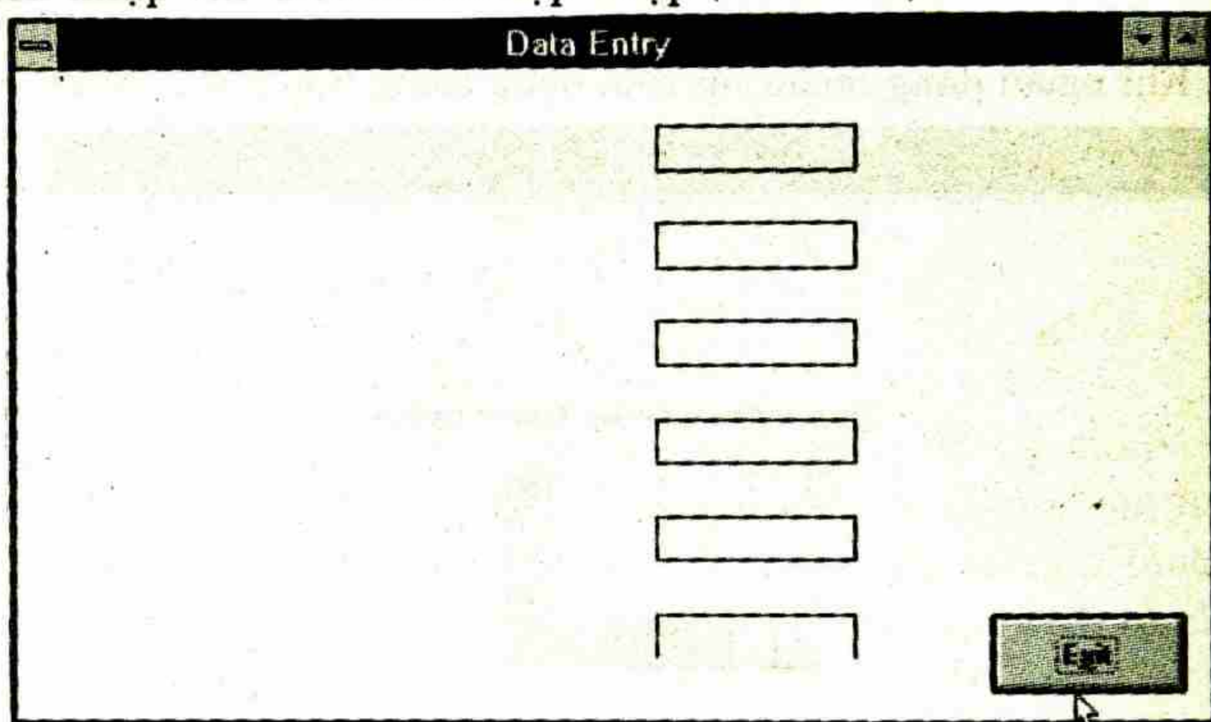
Bằng cách thiết đặt thuộc tính MultiLine và ScrollBars, bạn có thể dùng các hộp nhập nhiều dòng trong trình ứng dụng của mình. Các hộp nhập nhiều dòng cho phép nhận dữ liệu nhập vào của người dùng trong nhiều dòng.

SỬ DỤNG TIÊU ĐIỂM ĐỂ KIỂM SOÁT HỘP NHẬP

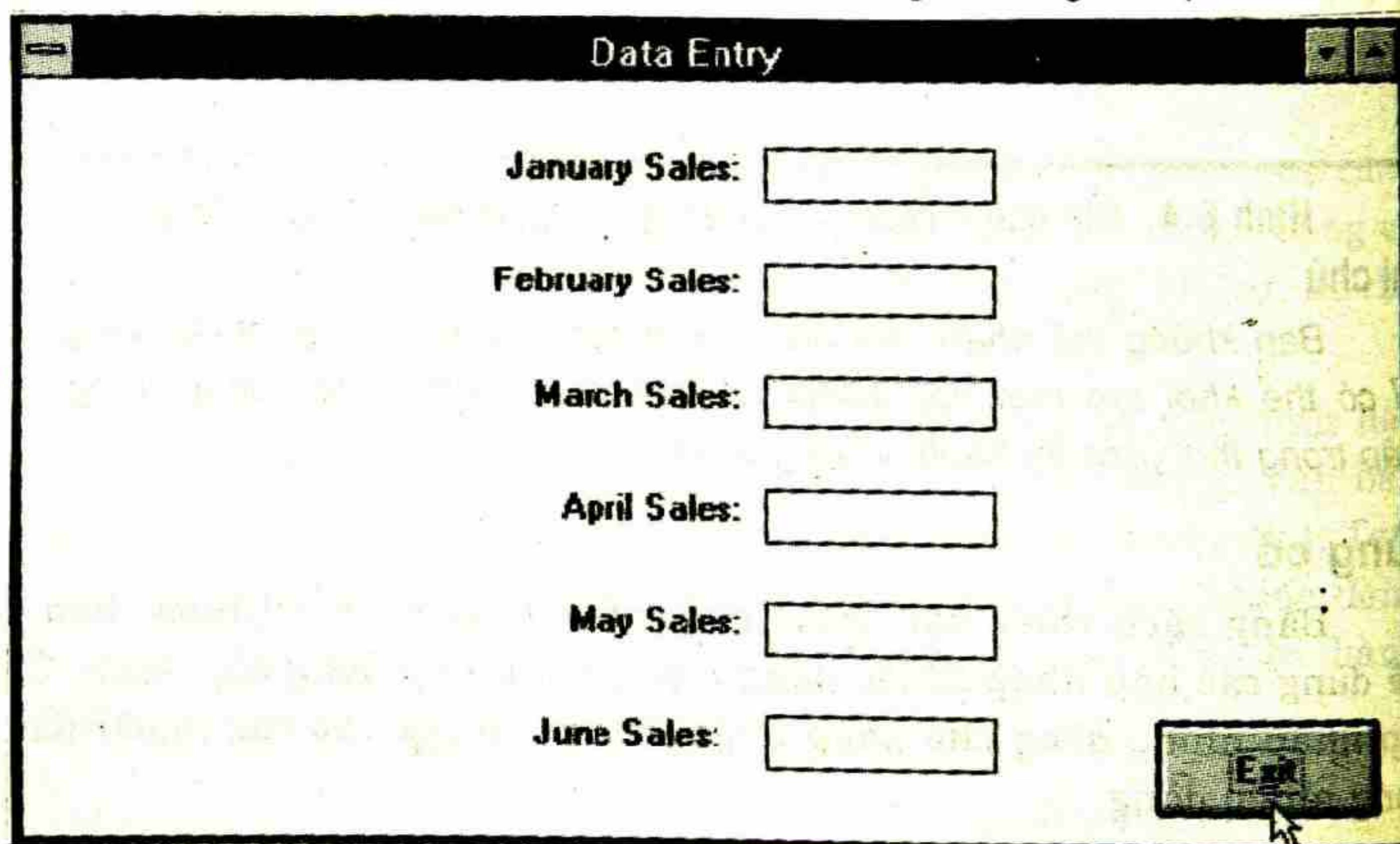
Khái niệm

Mặc dù các phím truy xuất không có hiệu lực với hộp nhập, bạn có thể dùng một mẹo nhỏ để cung cấp các phím truy xuất tới vùng nhập liệu của hộp nhập.

Khi bắt đầu thiết kế trình ứng dụng Visual Basic, bạn sẽ dùng các điều khiển hộp nhập để nhận dữ liệu từ người dùng. Đừng bỏ qua hộp nhập trên mẫu biểu và đợi người dùng cho biết những gì sẽ được nhập vào nó. Chẳng hạn, trong Hình 6.5, người dùng không biết dữ liệu nào nên nhập vào điều khiển hộp nhập (Text Box).



Hình 6.5. *Hộp nhập không có nhãn làm người dùng lẫn lộn.*



Hình 6.6. *Người dùng bây*

Bạn phải ghi nhãn hộp nhập với điều khiển nhãn và báo cho người dùng biết những gì bạn muốn. Chẳng hạn, trình ứng dụng trong Hình 6.6 sẽ xác định trình ứng dụng được mô tả trong Hình 6.5, nhưng các nhãn trước mỗi hộp nhập sẽ báo cho người dùng loại dữ liệu nhập vào.

Mách nước

Hãy bổ sung các phím truy xuất nhanh vào nhãn để mô tả nội dung dữ liệu của các hộp nhập.

Giả sử người dùng điền 6 ô lương. Anh ấy có thể muốn trở lại và thay đổi giá trị lương không phù hợp do lỗi nhập liệu. Khi bạn đặt hộp nhập trong một trình ứng dụng, hãy nghĩ cách cho người dùng cơ hội sửa các lỗi của họ bằng cách cung cấp phím truy xuất nhanh cho từng hộp nhập.

Như bạn biết, tổ hợp phím truy xuất nhanh là tổ hợp Alt+phím nhấn có thể áp dụng cho các điều khiển như nút lệnh. Trong Hình 6.6, người dùng có thể nhấn Tab để chuyển tới nút lệnh Exit, nhấp nút lệnh Exit, hay nhấn tổ hợp phím Alt+X – phím truy xuất nhanh của nút lệnh. Dấu gạch dưới sẽ cho biết ký tự nào đóng vai trò của phím truy xuất nhanh.

Điều khiển hộp nhập không có đề mục, cho nên bạn không thể thêm trực tiếp các phím truy xuất vào hộp nhập. Tuy nhiên, bạn có thể thêm các phím truy xuất vào đề mục nhãn. Ví dụ, giả sử bạn đã thay đổi nhãn đầu tiên trong Hình 6.6 từ January thành January. Phím truy xuất cho nhãn là tổ hợp phím Alt+J.

Nhưng hãy đợi một chút, có vấn đề ở đây. Nhãn không thể nhận tiêu điểm (focus)! Nếu nhãn có phím truy xuất và người dùng nhấn tổ hợp phím truy xuất, Visual Basic sẽ biết rằng tiêu điểm không thể chuyển tới nhãn. Nó sẽ gửi tiêu điểm đến điều khiển kế tiếp theo thứ tự TabIndex. Tất cả điều khiển đều có thuộc tính TabIndex. Mỗi điều khiển có giá trị thuộc tính TabIndex khác nhau. Thuộc tính TabIndex xác định thứ tự tiêu điểm. Giả sử bạn đã gán giá trị TabIndex bằng 0 cho nhãn January và giá trị TabIndex bằng 1 vào hộp nhập bên phải nhãn January. Khi người dùng nhấn tổ hợp phím Alt+J, tiêu điểm sẽ chuyển tới hộp nhập bởi vì điều khiển nhãn không thể nhận tiêu điểm.

Vì vậy, sau khi bạn đặt tất cả điều khiển lên mẫu biểu và định thuộc tính cho chúng, hãy quay trở lại từng điều khiển và đảm bảo rằng mỗi nhãn có một giá trị TabIndex nhỏ hơn 1 đơn vị so với điều khiển hộp nhập mà nhãn đó mô tả. Thứ tự tiêu điểm đảm bảo rằng tiêu điểm sẽ được chuyển từ điều khiển này sang điều khiển khác theo thứ tự bạn muốn khi người dùng nhấn Tab. Hình 6.7 mô tả 6 nội dung dữ liệu lương của 6 tháng và mỗi nhãn tương ứng có một phím truy xuất nhanh. Hình này còn cho biết giá trị TabIndex của từng điều khiển. Hãy định giá trị TabIndex sao cho tiêu điểm sẽ chuyển trực tiếp tới hộp nhập May khi người dùng nhấn tổ hợp phím Alt+Y.

Hình 6.7. *Thuôc tính TabIndex mô tả thứ tự phím truy xuất.*

Củng cố

Mặc dù bạn không thể truy xuất các phím vào hộp nhập, nhưng vẫn có thể bổ sung các phím truy xuất vào nhãn mô tả hộp nhập. Làm như thế, bạn sẽ giúp người dùng truy xuất nhanh hộp nhập bất kỳ trên mẫu biểu.

BIẾN CỐ ĐIỀU KHIỂN

Khái niệm

Bạn biết rằng khi người dùng nhấp nút lệnh và nhập văn bản vào trong hộp nhập, điều đó sẽ kích hoạt các biến cố mà chương trình của bạn có thể ngăn chặn. Phần này sẽ trình bày các biến cố có hiệu

lực của nút lệnh, nhãn và hộp nhập. Chương 4 sẽ chỉ cho bạn ngôn ngữ lập trình Visual Basic, để bạn hiểu biến cố nào có thể viết các thủ tục biến cố đáp ứng.

Sau đây là các bảng đầy đủ, giới thiệu với bạn tất cả biến cố có hiệu lực của những điều khiển bạn đã học. Trong chương kế tiếp, bạn sẽ bắt đầu lập trình. Mã lệnh bạn viết thường xuất hiện trong thủ tục biến cố. Bạn cần biết biến cố nào có hiệu lực, để có thể viết đúng các thủ tục biến cố.

Ghi chú

Khi học các điều khiển mới trong những chương kế tiếp, bạn sẽ thấy nhiều bảng thuộc tính và biến cố hơn. Thật thú vị!

Bảng 6.3 sẽ mô tả các biến cố liên quan đến mẫu biểu. Có lẽ biến cố mẫu biểu quan trọng nhất là biến cố Load, sẽ kích hoạt bất cứ khi nào người dùng chạy trình ứng dụng. Qua tập sách này, bạn sẽ dùng biến cố Load để đặt mã lệnh khởi động trong các trình ứng dụng sao cho mã lệnh khởi động thi hành ngay lập tức sau khi người dùng chạy trình ứng dụng và trước khi mẫu biểu xuất hiện trên màn hình.

Mách nước

Hãy nhớ rằng nếu bạn muốn xem biến cố nào có tác dụng với điều khiển xác định, hãy đặt điều khiển trên cửa sổ Form và nhấp đúp điều khiển. Visual Basic sẽ mở cửa sổ Code. Mở danh sách hộp combo xổ xuống bên phải để xem danh sách biến cố có hiệu lực cho điều khiển đó.

Bảng 6.3. *Các biến cố mẫu biểu*

Biến cố	Mô tả
Activate	Xảy ra khi mẫu biểu trở thành cửa sổ hoạt động. Trong Visual Basic, biến cố Activate xảy ra sau khi biến cố Load hiển thị mẫu biểu.
Click	Xảy ra khi người dùng nhấp mẫu biểu.
DblClick	Xảy ra khi người dùng nhấp đúp mẫu biểu.
Deactivate	Xảy ra khi mẫu biểu khác trở thành cửa sổ hoạt động.
DragDrop	Xảy ra khi thao tác kéo mẫu biểu hoàn thành.
DragOver	Xảy ra trong suốt thời gian diễn ra thao tác kéo mẫu biểu.
GotFocus	Xảy ra khi mẫu biểu nhận tiêu điểm.

KeyDown	Xảy ra khi người dùng nhấn phím và thuộc tính **KeyPreview** của điều khiển trên mẫu biểu được thiết đặt là **True**. Ngược lại, điều khiển nhận biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn phím trên mẫu biểu.
KeyUp	Xảy ra khi người dùng thả phím.
LinkClose	Xảy ra khi toán tử DDE dừng.
LinkError	Xảy ra khi toán tử DDE hỏng.
LinkExecute	Xảy ra khi toán tử DDE bắt đầu thi hành.
LinkOpen	Xảy ra khi toán tử DDE bắt đầu khởi động.
Load	Xảy ra khi mẫu biểu nạp và trước khi nó xuất hiện trên màn hình.
LostFocus	Xảy ra khi mẫu biểu mất tiêu điểm.
MouseDown	Xảy ra khi người dùng nhấn nút mouse trên mẫu biểu.
MouseMove	Xảy ra khi người dùng di chuyển mouse trên mẫu biểu.
MouseUp	Xảy ra khi người dùng thả mouse trên mẫu biểu.
Paint	Xảy ra khi Visual Basic phải vẽ lại mẫu biểu bởi vì đối tượng khác chồng lấp lên một phần mẫu biểu và sau đó người dùng di chuyển đối tượng và phơi bày ra phần bị che khuất của mẫu biểu.
QueryUnload	Xảy ra tức thời trước khi dừng trình ứng dụng.
Resize	Xảy ra khi người dùng định lại kích cỡ mẫu biểu.
Unload	Xảy ra khi mẫu biểu không được nạp bằng cách dùng lệnh Unload.

Lưu ý

Đừng quan tâm tới Bảng 6.3! Bạn sẽ sử dụng một vài biến cố trong suốt thời gian lập trình của mình.

Lưu ý rằng nội dung mô tả trong Bảng 6.3 rất nhiều từ *xảy ra*. Mỗi nội dung bảng này là một biến cố sẽ xảy ra như kết quả của hành động người dùng hay Windows. Vì vậy, nếu muốn làm điều gì khi người dùng nhấp mẫu biểu, bạn viết mã lệnh thi hành nhiệm vụ bạn muốn hoàn thành và đặt nó trong thủ tục biến cố Click của mẫu biểu. Nếu mẫu biểu tên MyForm, thủ tục biến cố Click sẽ có tên là MyForm_Click(), như bạn đã học trong Bài 4. Bạn sẽ bắt đầu viết mã lệnh cho các thủ tục biến cố trong bài tiếp theo.

Bảng 6.4 trình bày các biến cố có hiệu lực với điều khiển nút lệnh (Command Button) đặt trên mẫu biểu.

Bảng 6.4. *Các biến cố nút lệnh*

Biến cố	Mô tả
Click	Xảy ra khi người dùng nhấp nút lệnh.
DragDrop	Xảy ra khi thao tác kéo nút lệnh hoàn thành.
DragOver	Xảy ra trong suốt thời gian kéo nút lệnh.
GotFocus	Xảy ra khi nút lệnh nhận tiêu điểm.
KeyDown	Xảy ra khi người dùng nhấn phím và thuộc tính KeyPreview cho điều khiển bất kỳ trên mẫu biểu được thiết đặt là False. Ngược lại, mẫu biểu nhận biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn phím trên nút lệnh.
KeyUp	Xảy ra khi người dùng thả phím.
LostFocus	Xảy ra khi nút lệnh chuyển tiêu điểm sang điều khiển khác hoặc cho mẫu biểu.

Bảng 6.5 mô tả các biến cố có hiệu lực với những điều khiển nhãn (Label) bạn đặt trên mẫu biểu.

Bảng 6.5. *Các biến cố điều khiển nhãn*

Biến cố	Mô tả
Change	Xảy ra khi thuộc tính Caption của nhãn thay đổi.
Click	Xảy ra khi người dùng nhấp nhãn.
DblClick	Xảy ra khi người dùng nhấp đúp nhãn.
DragDrop	Xảy ra khi thao tác kéo nhãn hoàn thành.
DragOver	Xảy ra trong suốt quá trình kéo nhãn.
LinkClose	Xảy ra khi toán tử DDE dừng.
LinkError	Xảy ra khi toán tử DDE hỏng.
LinkNotify	Xảy ra khi toán tử DDE cho biết nhãn có một thông báo được thay đổi.
LinkOpen	Xảy ra khi toán tử DDE bắt đầu.
MouseDown	Xảy ra khi người dùng nhấn nút mouse trên nhãn.
MouseMove	Xảy ra khi người dùng di chuyển mouse trên nhãn.
MouseUp	Xảy ra khi người dùng thả nút mouse trên nhãn.

Lưu ý rằng không có biến cố GotFocus với nhãn bởi vì nhãn không bao giờ nhận tiêu điểm (focus). Bảng 6.6 mô tả các biến cố có hiệu lực với điều khiển hộp nhập (Text Box) bạn đặt trên mẫu biểu.

Bảng 6.6. *Các biến cố điều khiển hộp nhập*

Biến cố	Mô tả
Change	Xảy ra khi thuộc tính Text của hộp nhập thay đổi.
DragDrop	Xảy ra khi thao tác kéo hộp nhập hoàn thành.
DragOver	Xảy ra trong thời gian kéo hộp nhập.
GotFocus	Xảy ra khi hộp nhập nhận tiêu điểm.
KeyDown	Xảy ra khi người dùng nhấn phím và thuộc tính KeyPreview của điều khiển trong mẫu biểu thiết đặt là True. Ngược lại, mẫu biểu nhận biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn phím trên hộp nhập.
KeyUp	Xảy ra khi người dùng thả phím trên hộp nhập.
LinkClose	Xảy ra khi toán tử DDE dừng.
LinkError	Xảy ra khi toán tử hỏng.
LinkNotify	Xảy ra khi toán tử DDE thông báo cho hộp nhập một thông báo đã thay đổi.
LinkOpen	Xảy ra khi toán tử DDE bắt đầu.
LostFocus	Xảy ra khi hộp nhập chuyển tiêu điểm sang đối tượng khác.

Củng cố

Mỗi điều khiển có một bộ các thuộc tính và nút lệnh riêng. Các bảng trong bài này trình bày đầy đủ về mẫu biểu, nút lệnh, nhãn và hộp nhập. Bây giờ bạn đã biết cách làm việc với bốn đối tượng Visual Basic cơ bản này. Bài tiếp theo sẽ chỉ cho bạn cách thêm mã lệnh vào thủ tục biến cố.

Bài tập

Kiến thức tổng quát

1. Mẫu biểu có phải là một đối tượng không?
2. Đúng hay Sai: Các điều khiển nhãn có thể có nhiều hơn một dòng văn bản khi bạn thiết đặt thuộc tính MultiLine.
3. Pixel là gì?
4. Thuộc tính mẫu biểu nào xác định cách mẫu biểu hiển thị khi người dùng thấy mẫu biểu lần đầu tiên?
5. *Truncate* có nghĩa là gì?
6. Đúng hay Sai: Giả sử bạn muốn lưu nhiều văn bản trong nhãn. Bạn phải tạo một nhãn đủ lớn để lưu nội dung văn bản sao cho người dùng có thể thấy tất cả văn bản.

7. Đúng hay Sai: Bạn có thể thêm cả hai thanh cuốn ngang và dọc vào một nhãn.
8. Cách bạn có thể đảm bảo nhãn không được mở rộng để hiển thị một giá trị dài?
9. Đúng hay Sai: Bạn có thể nhập vào một giá trị văn bản nhiều dòng ban đầu bằng cách dùng cửa sổ Properties.
10. Đúng hay Sai: Nhãn có thể nhận tiêu điểm ngay khi bạn cung cấp một phím truy xuất nhanh cho nhãn.
11. Thuộc tính nào sẽ xác định thứ tự tiêu điểm?
12. Biến cố mẫu biểu quan trọng nhất là gì?
13. Biến cố mẫu biểu nào sẽ xảy ra trước: Load hay Activate?
14. Cửa sổ nào bạn có thể sử dụng để xem tất cả biến cố của cát đối tượng?
15. Tại sao không có biến cố GotFocus cho nhãn?

Lập trình...

16. Giả sử bạn đang viết thủ tục biến cố Change cho hộp nhập tên txtLastName. Tên thủ tục biến cố là gì?
17. Khi người dùng nhấn phím, hoặc mẫu biểu nhận phím nhấn hoặc điều khiển nhận phím nhấn, thuộc tính nào xác định đối tượng sẽ nhận phím nhấn?
18. Giả sử bạn cần thay đổi thanh tiêu đề mẫu biểu. Thuộc tính nào bạn sẽ thay đổi – thuộc tính Caption hay thuộc tính Name?
19. Hãy mô tả cách bạn thêm các phím truy xuất vào hộp nhập.

Tìm lỗi kỹ thuật

1. Tại sao bạn nên tránh đặt quá nhiều nhãn tự định kích cỡ trên mẫu biểu?
2. Điều gì sẽ xảy ra nếu bạn định thuộc tính AutoSize là True trước khi định thuộc tính WordWrap là True?

Phần nâng cao

Bạn muốn nhập các xác lập cửa sổ Property theo đơn vị inch thay vì twip, đơn vị đo mặc định. Cách bạn thay đổi đơn vị đo thành inch?

Bài thực hành 3

Bắt đầu công việc kinh doanh

Tóm tắt

Trong chương này, bạn đã tìm hiểu các thuộc tính và biến cố có thể có của mẫu biểu, nhãn, hộp nhập và nút lệnh. Visual Basic có thể xử lý thực sự những yêu cầu trình ứng dụng cần. Bạn đã nắm bắt:

- Cách đặt và di chuyển tiêu điểm (focus) từ đối tượng này đến đối tượng khác
- Giá trị thuộc tính cho từng đối tượng của trình ứng dụng
- Biến cố có tác dụng của từng đối tượng Visual Basic
- Lý do đôi khi phải quản lý nhãn và hộp nhập bằng cách dùng các thuộc tính thanh cuốn và định lại kích cỡ

Ứng dụng thực hành là PROJECT3.VBP. Bạn nên thiết kế, nạp và chạy trình ứng dụng để có thể theo dõi nội dung mô tả của bài thực hành này.

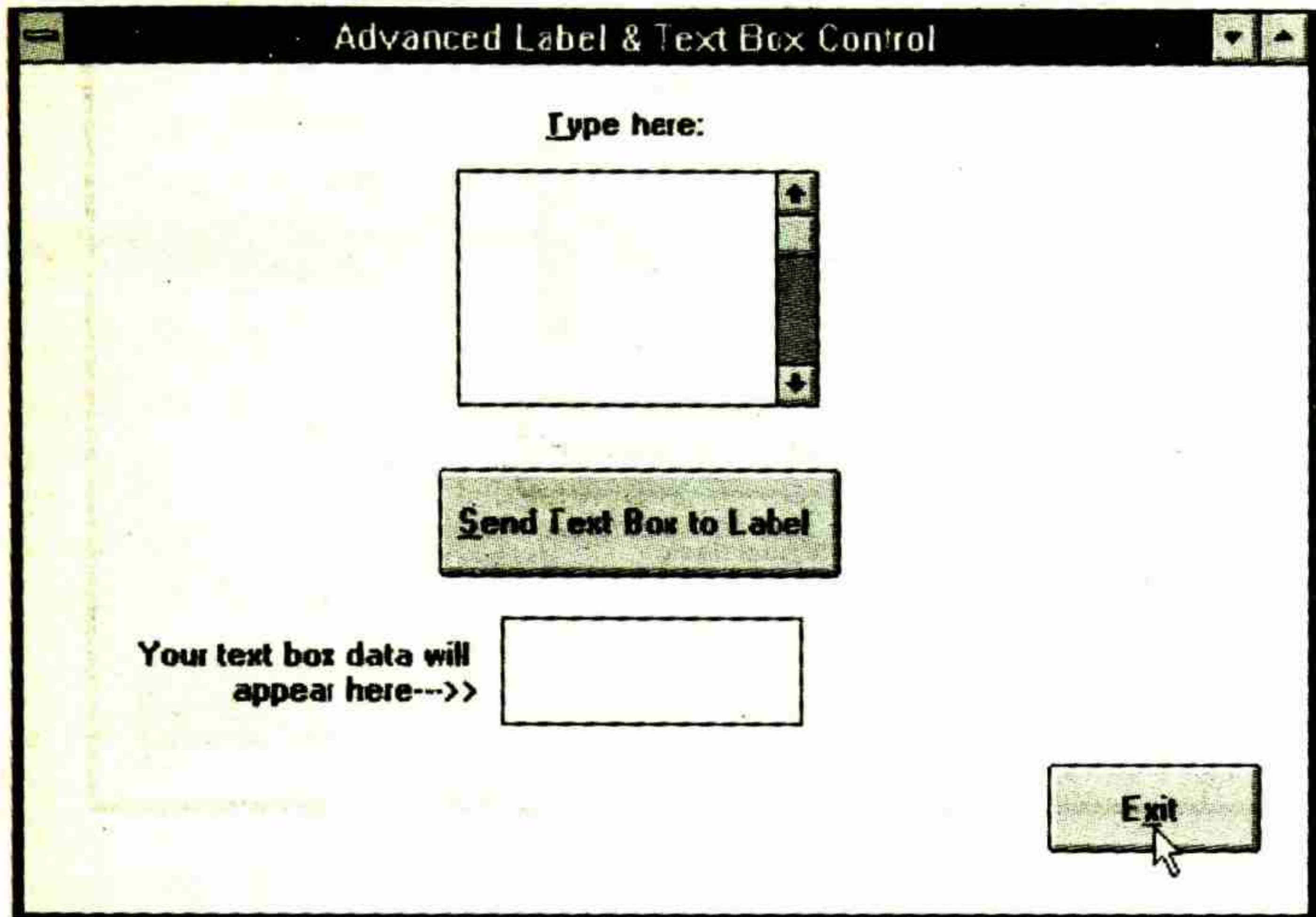
Mô tả chương trình

Hình P3.1 mô tả cách ứng dụng PROJECT3.VBP được nhìn thấy khi chạy chương trình. Chương trình gồm các đối tượng sau:

- Ba nhãn
- Một hộp nhập có thanh cuốn dọc
- Hai nút lệnh, mỗi nút có phím truy xuất riêng của nó

Lưu ý rằng nhãn trên cùng cũng có một phím truy xuất là tổ hợp phím Alt+T, sẽ gửi tiêu điểm tới hộp nhập.

Mục đích của chương trình này đòi hỏi một thông báo mà bạn có thể nhập vào trong hộp nhập. Thông báo có thể dài hay ngắn. Sau khi nhập thông báo, nhấp nút lệnh ở giữa hay nhấn tổ hợp phím Alt+S để gửi thông báo hộp nhập vào nhãn trống dưới nút lệnh giữa.



Hình P3.1. *Màn hình khi bắt đầu chương trình.*

Nhãn nhận dữ liệu của hộp nhập có thể tự điều chỉnh kích cỡ cho phù hợp với kích cỡ văn bản. Vì vậy, nếu bạn chỉ nhập một từ đơn, nhãn sẽ co lại cho vừa đủ một từ. Ngược lại, nếu bạn nhập văn bản nhiều dòng như một bài thơ, tất cả các dòng của văn bản sẽ xuất hiện trong nhãn bởi vì nhãn sẽ mở rộng để lưu tất cả.

Hoạt động chương trình

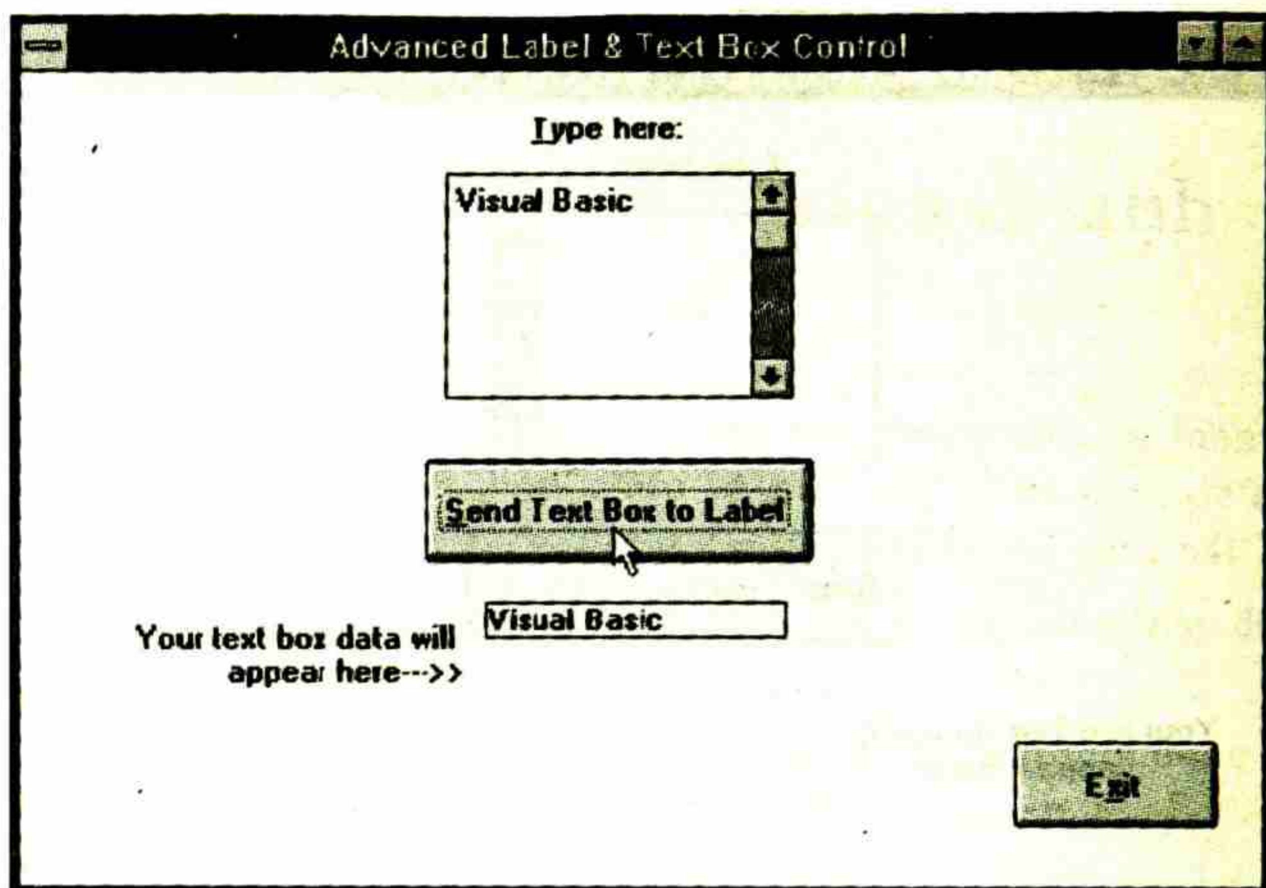
Bây giờ nhập từ **Visual Basic** vào hộp nhập. Nhấn tổ hợp phím Alt+S để gửi nội dung hộp nhập cho nhãn ở cuối màn hình. Nhãn sẽ co lại để lưu vừa đủ hai từ, như minh họa qua Hình P3.2.

Rõ ràng là nút lệnh sẽ kích hoạt biến cố giữ tất cả văn bản trong hộp nhập và gửi bản sao của nó tới nhãn ở cuối màn hình.

Nhãn có thể điều chỉnh kích cỡ ở cuối màn hình sở hữu những xác lập trong cửa sổ Property như sau:

AutoSize: True

WordWrap: True



Hình P3.2. *Xem cách nhãn định lại kích cỡ.*

Nếu thuộc tính AutoSize không thiết đặt là True, nhãn không tự co lại cho phù hợp giá trị văn bản nhỏ gán cho nhãn. Tương tự, khi bạn nhập một thông báo dài hơn vào hộp nhập và nhấn Alt+S để gửi nó tới nhãn phía dưới, nhãn sẽ không mở rộng cho phù hợp với thông báo dài. Giá trị WordWrap bằng True đảm bảo rằng nhãn mở rộng theo chiều dọc thay vì chiều ngang như nhãn dài một dòng.

Hãy thử một thông báo khác. Nhấn tổ hợp phím Alt+T để chuyển tiêu điểm trở lại hộp nhập. Nhấn phím Backspace để xóa hai từ hiện hành trong hộp nhập. Nhập vào bài thơ hay sau được một nhà soạn nhạc thích nhạc giao hưởng viết cách đây hàng trăm năm.

Visual Basic is like music,

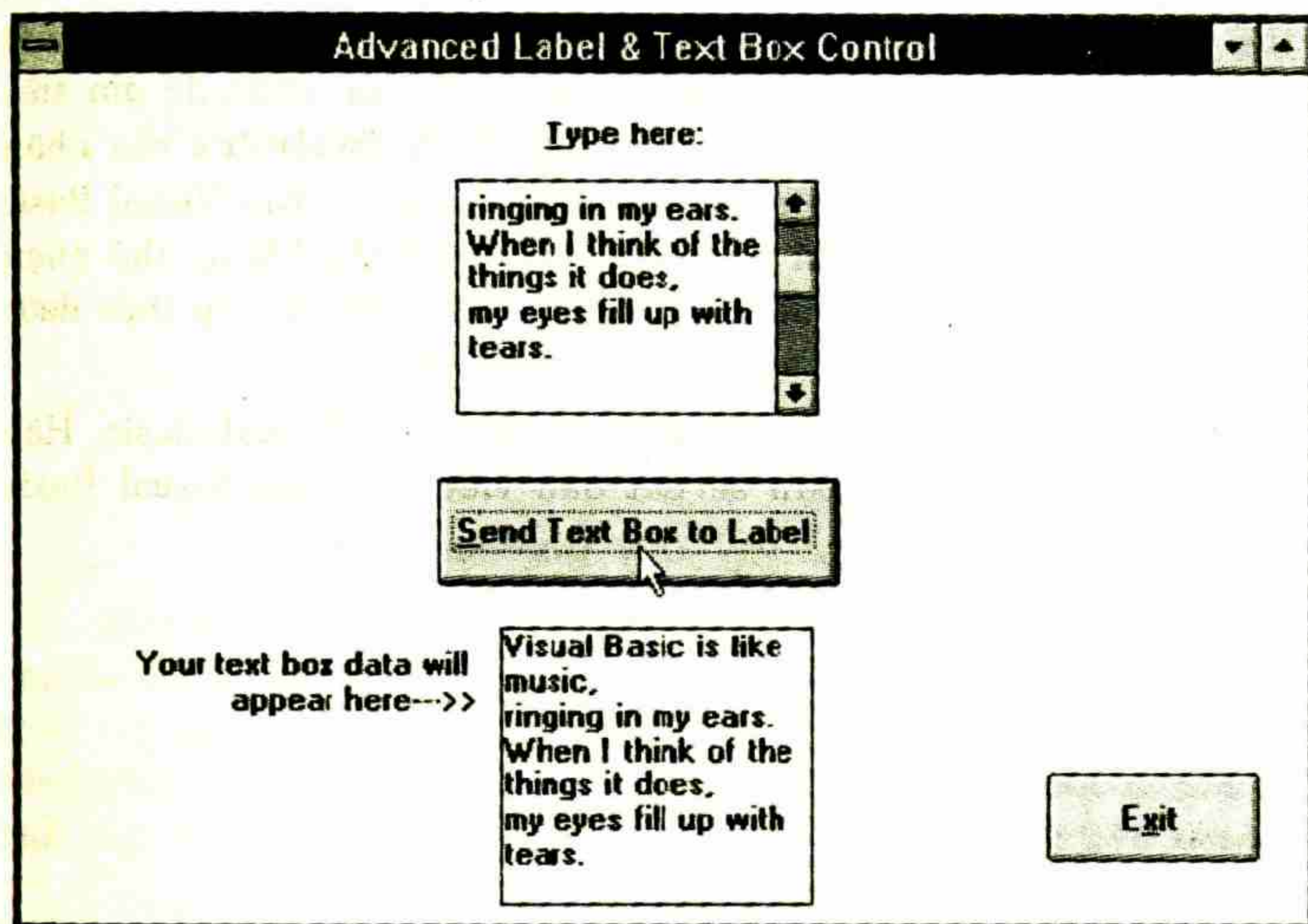
ringing in my ears.

When I think of the things it does,

my eyes fill up with tears.

Nhấn Alt+S để gửi bài thơ cho nhãn. Hãy xem cách nhãn mở rộng nhanh theo chiều dọc để lưu toàn bộ thông báo. Hình P3.3 minh họa những gì trông thấy trên màn hình của bạn.

```
***Begin Note***
***End Note***
***Begin Warning***
***End Warning***
```



Hình P3.3. *Nhãn trở nên lớn khi cần thiết.*

Ghi chú

Nếu bạn đã nhấn Enter ở cuối mỗi dòng bài thơ, màn hình của bạn sẽ tương tự màn hình minh họa trong Hình P3.3. Ngược lại, bài thơ của bạn sẽ bị dồn lại một chút so với cách nó xuất hiện trong Hình P3.3.

Mách nước

Mã lệnh các thủ tục biến cố trong chương trình thực hành Project3.vbp.

```
Private Sub cmdExit_Click()
    End
End Sub

Private Sub cmdSend_Click()
    lblLabel2.Caption = txtText1.Text
End Sub
```

Đóng trình ứng dụng

Trước bài này, bạn không thể đặt một nhãn có thể điều chỉnh kích cỡ trong thời gian thi hành cho phù hợp với văn bản nhận được. Hãy nhìn lại màn hình. Bạn sẽ thấy một mẹo nhỏ khác trong bài này: có thể gửi tiêu điểm tới hộp nhập bằng cách nhấn phím truy xuất (Alt+T) cho nhãn ở ngay trên hộp nhập. Cách duy nhất để gửi tiêu điểm tới hộp nhập là phải chắc chắn thuộc tính **TabIndex** của nhãn nhỏ hơn thuộc tính **TabIndex** của hộp nhập 1 đơn vị. Khi Visual Basic nhận biết phím truy xuất gửi tiêu điểm tới một nhãn không thể nhận tiêu điểm, Visual Basic sẽ gửi tiêu điểm tới điều khiển tiếp theo được xác định theo thứ tự giá trị TabIndex của đối tượng.

Bây giờ bạn có thể thoát trình ứng dụng và Visual Basic. Hãy nghỉ ngơi giây lát bởi vì bạn sẽ bắt đầu viết mã lệnh Visual Basic thực sự.

Chương IV

Bài 7

Biến, điều khiển và phép toán

- Các kiểu dữ liệu
- Biến lưu trữ
- Gán giá trị cho biến
- Biểu thức toán học
- Hàm Val()

Đây là lúc học ngôn ngữ lập trình Visual Basic. Cho đến bây giờ, bạn đã tạo các chương trình bằng cách đặt điều khiển trên mẫu biểu và thiết lập giá trị thuộc tính cho chúng. Lập trình Visual là trái tim của Visual Basic, và tính chất nhìn thấy được của Visual Basic là những gì phân biệt nó với nhiều ngôn ngữ lập trình truyền thống dựa trên văn bản.

Trong bài học này, còn có nhiều điều thú vị hơn về Visual Basic! Nếu học ngôn ngữ Visual Basic dựa trên văn bản và tương tác với điều khiển, bạn có thể thiết kế trình ứng dụng hoàn hảo. Ngôn ngữ lập trình Visual Basic – mã lệnh – thường phối hợp các điều khiển với nhau sao cho có thể phân tích và tính toán dựa trên dữ liệu

Cảnh báo

Visual Basic là ngôn ngữ nhưng không thực sự giống ngôn ngữ. Bạn sẽ thấy từ vựng của nó bao gồm những từ đơn giản mà bạn đã làm quen.

CÁC KIỂU DỮ LIỆU

Khái niệm

Ngôn ngữ Visual Basic làm việc với mọi loại dữ liệu. Trước khi học cách sử dụng dữ liệu, bạn sẽ học cách phân biệt giữa các kiểu dữ liệu khác nhau.

Visual Basic áp dụng và phân tích bảy kiểu dữ liệu. Bảng 7.1 mô tả những kiểu dữ liệu mà Visual Basic hỗ trợ. Một số dữ liệu bao gồm nhiều loại khác nữa. Ví dụ, tổng số đô–la có thể duy trì với cả kiểu *dữ liệu tiền tệ* (currency) lẫn kiểu dữ liệu đơn. Khi viết chương trình, bạn cần quyết định kiểu dữ liệu nào phù hợp nhất với giá trị dữ liệu trong chương trình.

Bảng 7.1. *Các kiểu dữ liệu Visual Basic*

Kiểu dữ liệu	**Mô tả**
Integer	Các giá trị số nguyên từ –32.768 đến 32.767.
Long	Các giá trị số nguyên với dãy số vượt quá các giá trị dữ liệu số nguyên. Giá trị dài từ –2.147.483.648 đến 2.147.483.647. Giá trị dữ liệu này chiếm nhiều không gian lưu trữ hơn giá trị số nguyên và ít hiệu quả hơn. Kiểu dữ liệu dài thường được gọi là số nguyên dài.
Single	Là các giá trị số từ –3.402823E+38 đến 3.402823E+38. Kiểu dữ liệu đơn thường được gọi là độ chính xác đơn.
Double	Các giá trị số từ –1.79769313486232E+308 đến 1.79769313486232E+308. Kiểu dữ liệu kép thường được gọi là độ chính xác kép.
Currency	Dữ liệu chứa số tiền đô từ –$922.337.203.685.477,5808 đến $922.337.203.685.477,5807.
String	Chuỗi dữ liệu từ 0 tới 65.500 ký tự chữ số. Chữ số nghĩa là giá trị dữ liệu có thể chứa chuỗi ký tự lẫn chuỗi số, thậm chí có thể chứa ký tự đặc biệt như ^%@.
Variant	Được dùng cho dữ liệu lưu trữ trong điều khiển và cho các giá trị ngày giờ.

Những giá trị dữ liệu sau có thể xem là kiểu dữ liệu số nguyên:

21

0

–9455

32766

Bạn cũng có thể lưu kiểu dữ liệu nguyên này như kiểu dữ liệu nguyên dài. Tuy nhiên việc làm này sẽ chiếm nhiều không gian lưu trữ và thời gian trừ khi bạn có kế hoạch thay đổi giá trị sau này thành số nguyên vô cùng lớn hoặc số nguyên nhỏ nhưng đòi hỏi kiểu dữ liệu.

Những giá trị dữ liệu sau phải được lưu trong kiểu dữ liệu nguyên dài:

32768

–95445

492848559

Những giá trị dữ liệu sau có thể lưu trong kiểu dữ liệu đơn:

0.01

565.32

–192.3424

9543.5645

6.5440E–24

Dĩ nhiên, bạn có thể lưu giá trị dữ liệu này ở dạng kép. Sử dụng kiểu dữ liệu kép chỉ khi bạn cần lưu giá trị cực lớn hoặc cực nhỏ.

Các giá trị dữ liệu dưới đây có thể lưu trong kiểu dữ liệu kép:

–5968.5765934211133

4.532112E+92

928374.344838273567899990

E đóng vai trò gì trong các số đi kèm? E là ký tự viết tắt của từ *exponent*. Trước đây, những nhà toán học cảm thấy mệt mỏi với việc viết số nguyên dài. Họ đã đưa ra ký hiệu ngắn được gọi là ký hiệu khoa học. Visual Basic sẽ hỗ trợ ký hiệu khoa học. Bạn có thể sử dụng ký hiệu khoa học cho giá trị số đơn và số kép. Khi sử dụng ký hiệu khoa học, bạn không phải viết hay nhập các số nguyên dài vô cùng lớn, trong trường hợp viết chương trình.

Khi một giá trị số đơn hay số kép có chứa ký tự E, theo sau là số, số đó đã được viết theo ký hiệu khoa học. Để chuyển một số được viết dưới dạng ký hiệu khoa học thành dạng đầy đủ, nhân số bên trái ký tự E với 10 lũy thừa của số bên phải E. Vì vậy, 2.9876E+17 nghĩa là nhân 2.9876 với 10 lũy thừa 17, là một số rất lớn – đó là số 10 theo sau 16 số 0. Tóm lại 2.9876E+17 cũng có nghĩa 29876 và theo sau 16 số 0.

Nếu sử dụng số Exponent âm, bạn hãy nhân số bên trái của E với 10 lũy thừa âm. Vì vậy, 2.9876E–17 bằng 2.9876 nhân với 10^{-17} – hoặc lấy 2.9876 chia cho 10^{17}– kết quả thực là một số vô cùng nhỏ.

Ghi chú

Không nên tự cho mình là nhà toán học? Điều đó đúng thôi – Visual Basic sẽ tính tất cả các phép toán cho bạn khi bạn học cách viết chương trình trong sách này. Thông thường, chỉ lập trình viên khoa học và kỹ sư mới sử dụng ký hiệu khoa học, và họ cảm thấy việc dùng nó thoải mái như đang ở nhà. Bảng 7.1 dùng ký hiệu khoa học vì không có cách hiển thị các giá trị dữ liệu kép nhỏ và lớn chứa 307 số 0.

Những giá trị dữ liệu sau có thể lưu ở kiểu dữ liệu currency:

```
123.45
0.69
63456.75
–1924.57
```

Kiểu dữ liệu currency có thể chấp nhận và tìm giá trị có đến bốn chữ số thập phân. Tuy nhiên, thông thường bạn lưu giá trị đôla và cent ở dạng currency, và giá trị này chỉ đòi hỏi hai chữ số thập phân.

Ghi chú

Visual Basic dùng những xác lập Windows International – được tìm thấy trong Control Panel nếu bạn muốn dành thời gian để xem – hầu tìm giá trị như ngày, giờ và currency. Vì vậy, giả như bạn khởi động Windows với nước Pháp, vốn sử dụng dấu phẩy thay vì dấu chấm thập phân như ở Mỹ, Visual Basic sẽ thay bằng dấu phẩy khi hiển thị giá trị.

Đừng bao giờ sử dụng giá trị dữ liệu currency đi kèm dấu $. Nói cách khác, dữ liệu sau không thể là giá trị currency ngay cả khi nó thuộc kiểu tương tự:

```
$5,234.56
```

Visual Basic không muốn bạn dùng dấu $ hay dấu phẩy trong dữ liệu số ở bất kỳ dạng nào – dĩ nhiên, trừ khi sử dụng dấu phẩy cho các số lẻ thập phân. Dữ liệu không thể chứa bất cứ ký tự nào ngoài chữ số, dấu (+), dấu (–) hoặc ký tự E (*exponent*). Visual Basic không thể xử lý dữ liệu nếu nó là số. Vì vậy, Visual Basic xử lý dữ liệu ở dạng chuỗi.

Các giá trị dữ liệu sau phù hợp với kiểu dữ liệu chuỗi:

"London Bridge"
"1932 Sycamore Street"
"^%#@#$%3939$%^&^&"

Hãy chú ý cặp dấu ngoặc kép bao quanh ba giá trị chuỗi. Giá trị chuỗi đòi hỏi dấu ngoặc kép thông báo cho Visual Basic vị trí chuỗi bắt đầu và kết thúc. Trường hợp bạn muốn chèn khoảng trống ở đầu hay cuối chuỗi, hãy chèn nó bên trong dấu ngoặc kép. Ví dụ, đây là ba chuỗi khác nhau, mỗi chuỗi ứng với một khoảng trống được chèn khác nhau: " house", "house ", " house ". Bạn không thể chèn trực tiếp dấu ngoặc kép bên trong chuỗi; Visual Basic sẽ cho rằng chuỗi được kết thúc ngay khi nó gặp dấu ngoặc kép thứ hai, mặc dù chưa thực sự kết thúc.

Ghi chú

Bạn vẫn còn hoài nghi. Thực ra có thể chèn dấu ngoặc kép bên trong chuỗi dữ liệu, nhưng sẽ làm lộn xộn vì thế cần áp dụng hàm thiết kế sẵn mà bạn sẽ học trong Bài 7.

Các giá trị dữ liệu sau có thể nhận kiểu dữ liệu Variant:

"^%#@#$%3939$%^&^&"
123.45
4.532112E+92
21
03–Mar–1996

Những giá trị này trông rất quen phải không? Kiểu dữ liệu Variant có thể chứa loại dữ liệu bất kỳ. Hãy nghĩ về loại dữ liệu được lưu trong điều khiển nhãn. Bạn thường muốn hiển thị số, số tiền, và ngày giờ trong nhãn trên mẫu biểu. Bạn có thể lưu dữ liệu Variant trong điều khiển. Dữ liệu từ điều khiển này cũng là Variant.

Visual Basic cung cấp cho bạn nhiều phương pháp chuyển đổi dữ liệu từ kiểu này thành kiểu khác. Ví dụ, "34" là giá trị chuỗi vì có dấu ngoặc kép. Bạn không thể thực hiện tính toán trên dữ liệu chuỗi. Tuy nhiên, nếu bạn lưu chuỗi có chứa một số hợp lệ và quyết định bạn cần tính giá trị, Visual Basic cung cấp thủ tục con chuyển đổi dữ liệu xây dựng sẵn cho phép biến đổi chuỗi thành số để bạn có thể thực hiện phép tính với giá trị của nó. Quá trình phân biệt các kiểu dữ kiệu sao cho có thể lưu trữ và mô tả giá trị dữ liệu chính xác.

Ghi chú

Visual Basic sử dụng kiểu dữ liệu Variant cho giá trị ngày giờ, như bạn sẽ học trong Chương 7.

Khái niệm mới

Hằng số (constant) là giá trị dữ liệu không thay đổi.

Giá trị dữ liệu mà bạn nhìn thấy trong mục này tất cả đều là hằng số. Hằng số không thay đổi. Ví dụ, số 7 thì luôn luôn là 7, và không có bất kỳ ý nghĩa nào ngoại trừ 7. Đôi khi bạn phải định chính xác loại hằng số cho giá trị dữ liệu để Visual Basic làm việc đúng với dữ liệu. Ví dụ, 7 là hằng số, và cả bạn lẫn Visual Basic có thể thắc mắc nên lưu 7 ở dạng số nguyên hay số nguyên dài.

Dữ liệu Variant là loại dữ liệu duy nhất mà Visual Basic thay đổi khi cần, sao cho khớp với trạng thái dữ liệu. Nói cách khác, giả sử bạn lưu giá trị số nguyên dài trong vùng dữ liệu Variant, bạn có thể tính với giá trị Variant. Còn như bạn lưu chuỗi trong vùng dữ liệu Variant, bạn sẽ có khả năng thực hiện tính chuỗi trên dữ liệu.

Nếu gặp trường hợp yêu cầu kiểu dữ liệu hằng số đặc trưng, bạn không phải áp dụng kiểu dữ liệu Variant mặc định. Bạn có thể định rõ kiểu dữ liệu nào là hằng số xác định. Bằng cách thêm một trong những ký tự hậu tố kiểu dữ liệu liệt kê trong Bảng 7.2 vào dữ liệu hằng số, bạn sẽ đáp ứng kiểu dữ liệu khi sử dụng hằng số.

Bảng 7.2. *Các ký tự hậu tố kiểu dữ liệu*

Hậu tố	Kiểu dữ liệu	Ví dụ
&	Long	14&
!	Single	14!
#	Double	14#
@	currency	14@

Không phải lúc nào cũng cần ký tự hậu tố. Hầu hết, bạn có thể thực hiện bằng cách gán hằng số cho điều khiển và vùng dữ liệu mà không cần thêm ký tự hậu tố ở cuối tên. Nhớ rằng Visual Basic giả sử hằng dữ liệu là kiểu dữ liệu Variant trừ khi bạn sử dụng ký tự hậu tố để ghi chồng kiểu dữ liệu. Bởi vì dữ liệu Variant có thể đổi thành bất kỳ kiểu dữ liệu khác, nên bất cứ lúc nào bạn vẫn có thể sử dụng kiểu dữ liệu Variant mặc định.

Chương trình bao gồm dữ liệu thay đổi và dữ liệu hằng. Đôi lúc bạn phải xác định giá trị hằng, như là số tháng trong năm, số giờ trong ngày, hay tên của công ty. Lúc khác, bạn lại làm việc với giá trị thay đổi theo thời gian, như là lương của nhân viên, giá trị đầu tư. Mục tiếp theo sẽ thảo luận cách khởi tạo và sử dụng giá trị dữ liệu thay đổi theo thời gian.

Mách nước

Bạn đã từng nghe nói đến chuyển động quay tròn chưa? Nếu trước đây bạn chưa bao giờ lập trình – thì các kiểu dữ liệu quả là vấn đề rất khó bởi nó vốn tuân theo quy tắc và dễ làm bạn nản lòng. Rõ ràng bạn chưa hình dung được lý do sử dụng kiểu dữ liệu. Mục kế tiếp sẽ giải thích tại sao những kiểu dữ liệu này lại quan trọng như thế. Tuy bạn không đọc ngay lập tức, nhưng cũng không nên bỏ qua!

Củng cố

Toàn bộ dữ liệu Visual Basic thuộc một trong bảy kiểu dữ liệu. Kiểu dữ liệu sẽ dùng để làm việc và giới hạn giá trị. Các kiểu dữ liệu khác nhau cho phép bạn phân loại dữ liệu và thực hiện hàm định rõ trên từng loại dữ liệu đó.

BIẾN LƯU TRỮ

Khái niệm

Như bạn biết, máy tính có bộ nhớ. Bạn có thể lưu giá trị dữ liệu trong vùng nhớ mà chương trình Visual Basic sẽ hoạt động. Khi định nghĩa biến, bạn dành các vị trí trong bộ nhớ cho biến dữ liệu, tên biến đó, và quyết định kiểu dữ liệu nào mà biến sẽ lưu giữ.

Khái niệm mới

Biến (variable) là vùng lưu trữ được đặt tên để chứa dữ liệu sẽ thay đổi.

Trong suốt chương trình, bạn cần lưu dữ liệu trong vùng nhớ trong khi tính dữ liệu. Vùng nhớ được gọi là biến. Biến không giống như hằng, nó có thể thay đổi giá trị. Nói cách khác, bạn có thể lưu một số trong biến ở phần đầu chương trình và sau đó thay đổi số đó trong chương trình. Một số biến không thể thay đổi – ví dụ, trình ứng dụng không cần thay đổi biến – nhưng thường nội dung biến sẽ thay đổi.

Biến y hệt hộp trong bộ nhớ thường chứa giá trị dữ liệu. Biến có hai đặc điểm:

- Mỗi biến có một tên.
- Mỗi biến có thể chứa duy nhất một loại dữ liệu.

Những mục kế tiếp sẽ giải thích cách bạn có thể yêu cầu biến từ Visual Basic sao cho bạn nhận được vị trí để định giá trị dữ liệu bạn cần lưu trữ.

Định nghĩa về biến

Khái niệm mới

Để định nghĩa biến thì phải tạo và đặt tên cho biến.

Chương trình có thể có nhiều biến như ý muốn của bạn. Trước khi có thể sử dụng biến, bạn phải yêu cầu Visual Basic tạo biến bằng cách định nghĩa biến. Khi định nghĩa biến, chỉ thị cho Visual Basic cung cấp:

- Tên biến
- Kiểu dữ liệu của biến

Một khi bạn định nghĩa biến, biến luôn lưu giữ kiểu dữ liệu nguồn. Vì vậy, biến có độ chính xác đơn chỉ có thể lưu giữ giá trị chính xác đơn. Nếu bạn lưu một số nguyên trong biến chính xác đơn, Visual Basic sẽ đổi số nguyên thành số có độ chính xác đơn trước khi nó được lưu vào biến. Quá trình chuyển đổi như thế là bình thường và không gây nhiều phiền toái.

Lệnh Dim sẽ định nghĩa biến. Dùng lệnh Dim để cho Visual Basic biết:

- Bạn cần một biến
- Tên biến
- Loại biến bạn cần

Dim – viết tắt của *dimension* – là lệnh Visual Basic bạn viết trong cửa sổ Code của trình ứng dụng. Hễ khi nào học một lệnh mới, bạn cần biết kiểu của lệnh đó. Sau đây là kiểu của lệnh Dim:

```
Dim VarName AS DataType
```

Trong đó, VarName là tên mà bạn cung cấp. Khi Visual Basic thi hành lệnh Dim đang chạy, nó sẽ tạo biến trong bộ nhớ và gán tên bạn đã cung cấp phù hợp với VarName của lệnh. *DataType* là một trong bảy kiểu dữ liệu của Visual Basic đã liệt kê ở Bảng 7.1.

Mách nước

Hãy xem biến như điều khiển nhãn bên trong mà người dùng không nhìn thấy. Biến có tên và lưu dữ liệu giống như điều khiển nhãn chứa thuộc tính Name và lưu giữ chú thích. Tuy nhiên, biến cụ thể hơn điều khiển, vì những yêu cầu kiểu dữ liệu của biến.

Tên biến cũng được đặt theo các qui ước hệt như điều khiển. Trong Bài 4 bạn đã tìm hiểu qui ước đặt tên cho thuộc tính Name của điều khiển – Ví dụ, tên phải dài từ 1 đến 40 ký tự, không thể là một từ khóa, và v.v.

Luôn sử dụng lệnh Dim hầu định nghĩa biến trước khi dùng nó. Nếu tùy chọn Options Environment Require Variable Definition được định là Yes – như giá trị mặc định của nó – Visual Basic sẽ đưa ra thông báo lỗi bất cứ khi nào bạn sử dụng biến mà bạn không định nghĩa. Tùy chọn này giúp bạn tránh sai lỗi chính tả trong tên biến và lỗi trong chương trình.

Giả sử bạn đưa một phần chương trình cho lập trình viên Visual Basic khác để tiếp tục công việc. Nếu bạn đòi hỏi mọi biến phải được định nghĩa nhưng lại không tin tưởng vào xác lập Options Environment Require Variable Definition của các lập trình viên khác, chọn (General) từ danh sách Object xổ xuống cửa sổ Code và thêm câu lệnh đặc biệt sau:

```
Option Explicit
```

Quá trình thiết lập tùy chọn Options Environment không mấy quan trọng, lập trình viên khó lòng áp dụng biến trong chương trình mà không định nghĩa chúng chính xác trong lệnh Dim. Hơn nữa, cơ chế định nghĩa biến giúp loại bỏ lỗi kỹ thuật, cho nên bạn cần có thói quen đặt Option Explicit trong phân mục (General) của từng cửa sổ Code chứa chương trình.

Lệnh sau định nghĩa biến ProductTotal:

```
Dim ProductTotal As Currency
```

Từ lệnh Dim, bạn biết rằng biến lưu giữ dữ liệu currency và tên của nó là ProductTotal. Tất cả các lệnh Visual Basic và thủ tục con cài đặt sẵn bắt buộc ký tự hoa đầu tiên. Cho dù bạn không phải đặt tên biến với ký tự hoa, đa số lập trình viên Visual Basic đều thêm ký tự hoa vào giúp phân biệt với những ký tự khác. (Nhớ rằng bạn không thể sử dụng khoảng trống trong tên của biến.)

Ghi chú

Lệnh Static và Global cũng định nghĩa biến, như bạn sẽ bắt gặp trong Chương 6 và Chương 8.

Các câu lệnh sau định nghĩa biến số nguyên, biến có độ chính xác đơn và biến có độ chính xác kép:

```
Dim Length As Integer
Dim Price As Single
Dim Structure As Double
```

Lưu ý

Ngoại trừ số ít trường hợp mà bạn sẽ học trong Bài 8, bạn sẽ không bao giờ định nghĩa hai biến cùng tên. Visual Basic không biết biến nào đang được tham chiếu khi bạn sử dụng chúng.

Khái niệm mới

Variable-length string là chuỗi có độ dài thay đổi tùy ý.

Nếu bạn muốn viết chương trình lưu nội dung nhập vào hộp nhập tên người dùng, bạn sẽ định nghĩa chuỗi giống như sau:

```
Dim FirstName As String
```

Bạn có thể tưởng tượng khi định nghĩa chuỗi. Chuỗi FirstName có thể chứa chuỗi bất kỳ dài từ 0 đến 65.500 ký tự. Bạn sẽ tìm hiểu mục tiếp theo, cách lưu trữ dữ liệu trong chuỗi. FirstName có thể giữ dữ liệu hầu như ở bất kỳ kích cỡ nào. Bạn có thể lưu chuỗi ngắn trong FirstName như là "Joe" và sau đó lưu chuỗi dài hơn trong FirstName như "Mercedes" chẳng hạn. FirstName là chuỗi có độ dài thay đổi.

Khái niệm mới

Fixed-length string - Chuỗi có độ dài cố định - lưu giữ chuỗi có kích thước tối đa cố định.

Đôi khi bạn muốn giới hạn số lượng văn bản mà chuỗi sẽ lưu. Ví dụ, bạn có thể định nghĩa biến chuỗi nhằm lưu giữ tên mà bạn đọc từ tập tin trên đĩa. Sau đó, hiển thị nội dung của chuỗi dưới dạng nhãn trên mẫu biểu. Tuy nhiên, nhãn của mẫu biểu có chiều dài cố định, giả sử thuộc tính AutoSize ấn định ở True. Vì vậy, bạn muốn lưu giữ biến chuỗi với chiều dài hợp lý. Lệnh Dim sau đây sẽ minh họa cách bạn có thể thêm tùy chọn StringLength khi muốn định nghĩa chuỗi có độ dài cố định:

```
Dim Title As String * 20
```

Title là tên của biến chuỗi vốn có thể lưu giữ chuỗi từ 0 đến 20 ký tự. Giả như chương trình lưu giá trị chuỗi dài hơn 20 ký tự trong Title, Visual Basic sẽ tiếp nhận chuỗi và chỉ lưu 20 ký tự đầu tiên.

Sau đây là cách viết vắn tắt: bạn có thể bỏ qua mô tả As Variant khi định nghĩa biến. Lệnh Dim như sau:

```
Dim Value As Variant
```

thì chính xác như

```
Dim Value
```

Chuẩn mực hoàn hảo là tạo mã lệnh càng rõ ràng càng tốt. Nói cách khác, As Variant đòi hỏi nhập thêm từ bàn phím, nhưng sử dụng As Variant yêu cầu phải rõ ràng hơn biến Variant. Sau này khi muốn bảo trì chương trình, bạn sẽ biết rằng bạn muốn nói về một biến Variant và không sơ ý nhầm lẫn với kiểu dữ liệu khác.

Sử dụng biến để lưu giữ kết quả tính toán tức thời, cũng như các giá trị dữ liệu do người dùng nhập và những điều khiển chẳng hạn như hộp nhập. Biến làm việc xuất sắc cùng với điều khiển trên mẫu biểu. Cho đến bây giờ, bạn đã học cách định nghĩa biến nhưng không lưu dữ liệu trong biến. Phần kế tiếp sẽ giải thích cách thức lưu và nhận giá trị biến mà bạn định nghĩa.

Lưu ý

Trường hợp gọi biến hằng bằng tên, bạn phải dùng tên đó trong toàn bộ chương trình. Tên biến Sale không giống Sales. Còn như định nghĩa biến bằng cách sử dụng tên và sau đó chỉ thay đổi ký tự hoa nào đó, Visual Basic sẽ đổi ký tự hoa nguồn thành ký tự hoa mới. Nói cách khác, nếu bạn định nghĩa biến Profit *và dùng tên đó một lúc trong chương trình, nhưng sau đó chương trình lại gọi biến* PROFIT, *Visual Basic sẽ thay đổi mọi biến có tên* Profit *thành* PROFIT.

Visual Basic nhanh chóng hỗ trợ khi bạn cần định nghĩa vài biến. Thay vì liệt kê từng định nghĩa biến trên mỗi dòng như sau:

```
Dim A As Integer
Dim B As Double
Dim C As Integer
Dim D As String
Dim E As String
```

Bạn có thể tổ hợp những biến cùng kiểu dữ liệu trên một dòng. Ví dụ,

```
Dim A, C As Integer
Dim B As Double
Dim D, E As String
```

Một số lập trình viên dùng mã lệnh định nghĩa tất cả các biến trên cùng một dòng, cho dù có quá nhiều định nghĩa biến làm cho lệnh Dim khó quản lý. Ví dụ

```
Dim A, C As Integer, B As Double, Dim D, E As String.
```

Thay lệnh Dim bằng ký tự hậu tố của biến. Sẽ có tổ hợp ký tự hậu tố cho tên biến để xác định kiểu dữ liệu chứ không phải định nghĩa biến. Bảng 7.2 liệt kê các ký tự hậu tố dành cho những giá trị hằng. Các biến dùng trong những phiên bản hoàn chỉnh được nêu ra như sau:

Hậu tố	Kiểu dữ liệu	Ví dụ
%	integer	Age%
&	long	Amount&
!	single	Distance!
#	double	KelvinTemp#
@	currency	Pay@
$	string	LastName$

Câu lệnh Option Explicit không thể xuất hiện trong phân mục (General) của cửa sổ Code bởi vì những ký tự cuối không đủ để định nghĩa biến.

Biến LastName$ thuộc biến chuỗi, và Visual Basic biết biến này là biến chuỗi ngay cả khi lệnh Dim không định nghĩa biến dưới dạng chuỗi. Bạn sẽ không dùng thêm hậu tố biến bởi vì bạn sẽ viết chương trình rõ ràng hơn nếu sử dụng Dim. Tuy nhiên, bạn có thể nhập mã lệnh được viết bởi người khác có sử dụng ký tự hậu tố trong biến, và bạn nên chú trọng đến ý nghĩa của ký tự hậu tố.

GÁN GIÁ TRỊ CHO BIẾN

Khái niệm mới

Chuỗi null là chuỗi rỗng có chiều dài bằng 0 và thường được biểu thị trong dấu ngoặc kép (" ").

Khi bạn định nghĩa biến lần đầu tiên, Visual Basic sẽ gán giá trị 0 trong biến số và chuỗi rỗng thuộc biến chuỗi. Sử dụng câu lệnh gán khi bạn muốn đặt giá trị dữ liệu khác vào biến. Biến lưu giữ dữ liệu dưới dạng dữ liệu xác định, và nhiều dòng trong cửa sổ Code của chương trình Visual Basic gồm có câu lệnh gán dữ liệu cho biến. Sau đây là kiểu câu lệnh gán:

```
[Let]VarName = Expression
```

Tên lệnh Let là tùy chọn; hiện nay ít khi dùng đến nó. VarName là tên biến mà bạn định nghĩa bằng cách sử dụng câu lệnh Dim. Expression ở đây có thể là hằng, biến hoặc biểu thức toán học.

Giả sử bạn cần lưu giá trị 18 là độ tuổi nhỏ nhất (MinAge) vào biến số nguyên. Các câu lệnh gán sau có thể thực hiện điều đó:

```
Let MinAge = 18
```

Và

```
MinAge = 18
```

Ghi chú

Phần còn lại của tập sách này sẽ không đề cập đến từ Let trong các câu lệnh gán bởi lẽ từ này trở nên dư thừa.

Muốn lưu nhiệt độ trong biến chính xác đơn có tên TodayTemp, bạn có thể thực hiện như sau:

```
TodayTemp = 42.1
```

Kiểu dữ liệu của Expression phải phù hợp kiểu dữ liệu của biến mà bạn gán. Nói cách khác, câu lệnh sau không hợp lệ. Nó sẽ đưa ra lỗi trong chương trình Visual Basic nếu bạn cố dùng nó.

```
TodayTemp = "Forty-Two point One"
```

Đặt trường hợp TodayTemp là biến chính xác đơn, bạn không thể gán chuỗi cho nó. Tuy nhiên, Visual Basic thường thực hiện phép chuyển đổi nhanh cho bạn khi quá trình chuyển đổi đã chuẩn xác. Ví dụ, nó có khả năng thực hiện phép gán dưới đây ngay cả khi bạn đã định nghĩa Measure là biến chính xác kép:

```
measurement = 921.23
```

Chỉ thoáng nhìn, tưởng chừng 921.23 là số chính xác đơn vì kích cỡ của nó. 921.23 thực sự là giá trị dữ liệu Variant. Xin nhắc lại Visual Basic giả sử mọi hằng dữ liệu là Variant trừ khi bạn thêm ký tự hậu tố cụ thể vào hằng hầu tạo kiểu dữ liệu hằng khác. Visual Basic có thể chuyển đổi an toàn và dễ dàng giá trị Variant thành giá trị chính xác kép. Đó chính là những gì Visual Basic thực hiện, chính vì thế phép gán được tiến hành thông suốt.

Bổ sung vào hằng, bạn có thể gán biến khác cho biến. Hãy xem kỹ mã lệnh sau:

```
Dim Sales As Single, NewSales As Single
Sales = 3945.42
NewSales = Sales
```

Khi câu lệnh thứ ba hoàn thành, cả hai biến Sales và NewSales có giá trị hằng 3945.42.

Thật tự nhiên để gán biến cho điều khiển và ngược lại. Chẳng hạn, giả sử người dùng nhập giá trị 18.34 vào thuộc tính Text trong hộp nhập. Nếu thuộc tính Name của hộp nhập là txtFactor, câu lệnh dưới đây sẽ lưu giá trị của hộp nhập vào biến FactorVal:

```
FactorVal = txtFactor.Text
```

Giả sử bạn đã định nghĩa Title là biến chuỗi với chiều dài cố định bằng 10, nhưng người dùng nhập Mondays Always Feel Blue trong thuộc tính Text mà bạn muốn gán cho Title. Visual Basic chỉ lưu 10 ký tự đầu tiên của điều khiển vào Title và cắt bỏ phần còn lại của biến chuỗi. Vì vậy, Title chỉ giữ lại chuỗi "Mondays Al".

Tóm tắt

Đây là mã lệnh của vài chương trình đầu tiên ôn lại mà bạn sẽ tìm thấy trong phần còn lại của tập sách. Ví dụ 7.1 bao gồm thủ tục biến cố ngắn vốn gán chú thích của nút lệnh mới.

Củng cố

Bạn có thể làm dữ liệu xuất hiện ngay trên mẫu biểu bằng cách gán thuộc tính Text của các hộp nhập hoặc thuộc tính Caption của các nhãn và nút lệnh. Giả sử bạn đặt nút lệnh cmdJoke trên mẫu biểu. Thủ tục biến cố được trình bày trong Ví dụ 7.1 sẽ thay đổi chú thích khi người dùng nhấp nút lệnh.

Ví dụ 7.1. *Thủ tục biến cố gán chú thích cho nút lệnh mới*

```
1: Sub cmdJoke_Click ()
2: cmdJoke.Caption = "Bird Dogs Fly"
3: End Sub
```

Lưu ý

Xuyên suốt cuốn sách, bạn thường bắt gặp những ví dụ mã lệnh kèm theo số đặt bên trái. Sử dụng số hầu tham chiếu tới từng dòng. Không nên nhập số hoặc dấu hai chấm (:) theo sau số khi bạn nhập mã lệnh chương trình.

Phân tích

Không có vấn đề gì với thuộc tính Caption của nút lệnh được đặt ở phần đầu chương trình. Khi người dùng nhấp nút lệnh, thủ tục biến cố này sẽ thi hành và chú thích của nút lệnh sẽ đổi thành Bird Dogs Fly (dòng 2).

Mách nước

Bạn sẽ thấy chương trình toàn diện hơn để gán giá trị dữ liệu cho điều khiển trong bài thực hành của chương này.

Bạn có thể thêm tập tin CONSTANT.BAS định nghĩa tên hằng vào project để điều khiển thủ tục biến cố bên trong. Câu lệnh gán sau đây sẽ bổ sung đường viền đơn bao quanh nhãn lblSinger:

```
lblSinger.BorderStyle = FIXED_SINGLE
```

Hẳn nhiên, bạn phải biết hằng số có tên bên trong CONSTANT.BAS trước khi bạn có thể gán chúng. Qua tập sách này, bạn sẽ biết khi nào có thể sử dụng tên hằng cho việc thiết lập thuộc tính.

Ghi chú

Khi bạn không sử dụng CONSTANT.BAS, chỉ việc gán số trong trường hợp thuộc tính điều khiển có thể chấp nhận vùng giá trị có giới hạn. Lấy ví dụ, giá trị có thể xuất hiện khi bạn chọn thuộc tính BorderStyle cho nhãn trong cửa sổ Properties là 0–None và 1–Fixed Single. Để gán trực tiếp đường viền không sử dụng tên biến hằng, chỉ cần gán 0 hay 1. Không nên giải thích rõ ràng nội dung thuộc tính. Ví dụ:

```
lblSinger.BorderStyle = 1
```

Củng cố

Bằng cách dùng câu lệnh, bạn có thể gán hằng, tên hằng, giá trị điều khiển và biến cho các biến cũng như điều khiển khác. Câu lệnh gán đòi hỏi biến phải được định nghĩa trước bằng lệnh Dim nếu khoản mục (General) của chương trình chứa câu lệnh Option Explicit 1 như đã đề cập.

BIỂU THỨC TOÁN HỌC

Khái niệm

Giá trị dữ liệu và điều khiển không chỉ là những loại phép gán duy nhất mà bạn có thể thực hiện. Với các toán tử toán học của Visual Basic, bạn dễ dàng tính toán và gán kết quả biểu thức cho biến khi viết câu lệnh gán vốn chứa biểu thức.

Khái niệm mới

Toán tử (operator) là từ hay ký hiệu thực hiện phép tính và xử lý dữ liệu.

Thật dễ dàng để chỉ thị Visual Basic thực hiện phép tính. Bảng 7.3 mô tả những toán tử chính của Visual Basic. Bên cạnh đó còn có các toán tử khác, song những toán tử trong Bảng 7.3 vừa đủ cho phần lớn chương trình mà bạn viết. Hãy xem kỹ các toán tử. Bạn đã quen thuộc với hầu hết những toán tử này bởi vì chúng hoạt động và trông như bản sao thế giới thực.

Bảng 7.3. *Các toán tử tính toán chủ yếu*

Toán tử	Ví dụ	Diễn giải
+	Net + Disc	Cộng hai giá trị
–	Price – 4.00	Lấy giá trị này trừ đi giá trị kia
*	Total * Fact	Nhân hai giá trị
/	Tax / Adjust	Lấy số này chia cho số khác
^	Adjust ^ 3	Tăng giá trị cho lũy thừa
& or +	Name1 & Name2	Nối hai chuỗi

Bạn muốn lưu số tiền lương chênh lệch hàng năm (được lưu trong biến AnnualSales) và lương phải trả (được lưu trong biến CostOfSales) trong biến có tên NetSales. Giả sử có tất cả ba biến đã được định nghĩa và khởi tạo, câu lệnh gán sau sẽ tính toán giá trị đúng cho NetSales:

```
NetSales = AnnualSales – CostOfSales
```

Lệnh gán này ra chỉ thị Visual Basic tính giá trị biểu thức và lưu kết quả trong biến NetSales. Dĩ nhiên, bạn cũng có thể lưu kết quả biểu thức này trong thuộc tính Caption hoặc Text.

Nếu bạn muốn tăng lũy thừa lên một giá trị – nghĩa là nhân giá trị đó với chính nó số lần xác định, bạn có thể làm như thế. Mã lệnh sau sẽ gán 10000 cho Value bởi vì 10 lũy thừa 4 – 10×10×10×10 bằng 10,000:

```
Years = 4
Value = 10 ^ Years
```

Không nên lo lắng về những biểu thức phức tạp, Visual Basic tính toàn bộ kết quả trước khi lưu kết quả đó trong biến ở bên trái dấu bằng. Ví dụ, lệnh gán sau quả là dài, song Visual Basic sẽ tính kết quả và lưu giá trị trong biến Ans:

```
Ans = 8 * Factor – Pi + 12 * MonthlyAmts
```

Cơ chế kết hợp các biểu thức thường đem lại kết quả không thể hiểu được bởi vì Visual Basic sẽ tính kết quả toán học theo thứ tự ưu tiên. Visual Basic luôn tính lũy thừa trước nếu một hoặc nhiều toán tử ^ xuất hiện trong biểu thức. Sau đó,Visual Basic tính tất cả các phép nhân và chia trước khi thực hiện phép cộng và trừ.

Visual Basic gán 13 cho Result trong câu lệnh sau:

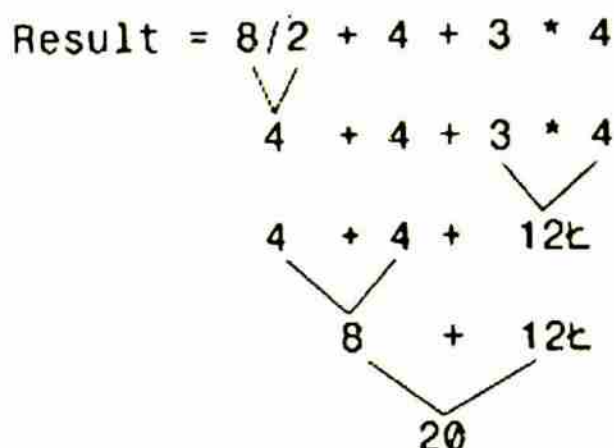
```
Result = 3 + 5 * 2
```

Thoạt đầu, bạn sẽ nghĩ Visual Basic sẽ gán 16 cho Result bởi vì 3 + 5 bằng 8 và 8 * 2 bằng 16. Tuy nhiên, theo quy tắc Visual Basic luôn tính phép nhân và chia nếu phép chia tồn tại trong biểu thức trước khi tính phép cộng. Vì vậy, đầu tiên Visual Basic tính giá trị 5 * 2 bằng 10, kế tiếp cộng 3 với 10 để được 13. Chỉ lúc đó nó mới gán 13 cho Result.

Trường hợp cả hai phép nhân và chia cùng xuất hiện trong biểu thức, Visual Basic sẽ tính từ trái sang phải. Ví dụ, Visual Basic gán 20 cho biểu thức sau:

```
Result = 8 / 2 + 4 + 3 * 4
```

Hình 7.1 cho thấy cách Visual Basic tính biểu thức này. Visual Basic sẽ tính phép chia đầu tiên, bởi lẽ phép chia xuất hiện bên trái phép nhân. Nếu phép nhân xuất hiện bên trái phép chia, Visual Basic sẽ thực hiện phép nhân trước. Sau khi Visual Basic tính toán các kết quả trung gian cho phép chia và phép nhân, nó sẽ thực hiện phép cộng và lưu kết quả cuối cùng là 20 vào Result.



Hình 7.1. *Visual Basic tính biểu thức theo thứ tự ưu tiên.*

Ghi chú

Thứ tự tính toán có nhiều tên. Lập trình viên thường sử dụng thuật ngữ thứ tự toán tử, độ ưu tiên của toán tử, hoặc cấp bậc phép toán.

Có thể bỏ qua tính ưu tiên toán tử bằng cách sử dụng cặp dấu ngoặc đơn (). Trong biểu thức, Visual Basic luôn tính giá trị trong cặp dấu ngoặc đơn trước, thậm chí nó bỏ qua cả toán tử ưu tiên. Câu lệnh gán dưới đây lưu 16 vào Result bởi vì cặp dấu ngoặc buộc Visual Basic tính phép cộng trước phép nhân:

```
Result = (3 + 5) * 2
```

Biểu thức sau đây lưu lũy thừa 1/5 của 125 cho biến root5:

```
root5 = 125 ^ (1/5)
```

Như bạn có thể thấy từ biểu thức này, Visual Basic hỗ trợ lũy thừa phân số.

Khái niệm mới

Concatenation nghĩa là tính kết nối hai hay nhiều chuỗi với nhau.

Một trong các toán tử chính của Visual Basic lại không làm gì với việc tính toán. Toán tử nối một chuỗi vào cuối chuỗi khác. Giả sử người dùng đã nhập tên trong điều khiển nhãn lblFirst và họ trong điều khiển lblLast. Biểu thức nối sau đây sẽ lưu họ tên đầy đủ trong biến chuỗi FullName:

```
FullName = lblFirst & lblLast
```

Mặc dù có vấn đề ở đây, nhưng không rõ ràng lắm – không có khoảng trống giữa hai tên. Toán tử & không tự động chèn khoảng trống bởi vì bạn không muốn khoảng trống khi nối hai chuỗi. Vì vậy, có lẽ bạn phải nối chuỗi thứ 3 giữa hai chuỗi này như sau

```
FullName = lblFirst & " " & lblLast
```

Visual Basic hỗ trợ toán tử đồng nghĩa, dấu +, với toán tử &. Nói cách khác, câu lệnh gán sau đây đồng nhất với câu lệnh trước:

```
FullName = lblFirst + " " + lblLast
```

Ngay cả Microsoft, người sáng tạo ra Visual Basic, đều cho rằng bạn dùng toán tử & để nối chuỗi và dự trữ toán tử + cho phép cộng các số. Cơ chế sử dụng cùng toán tử cho hai loại thao tác khác nhau có thể dẫn đến sự nhầm lẫn.

Củng cố

Các toán tử toán học cho phép bạn thực hiện mọi phép tính và phép gán. Visual Basic muốn nói là không bao giờ phải tái sử dụng một máy tính! Tuy nhiên, khi bạn áp dụng biểu thức, hãy nhớ đến tính ưu tiên toán tử. Nếu còn nghi ngờ, hãy dùng cặp dấu ngoặc đơn nhằm trình bày chính xác các phép toán trong biểu thức bạn muốn Visual Basic tính đầu tiên.

HÀM Val()

Khái niệm

Visual Basic hiểu toàn bộ dữ liệu lưu trong điều khiển trên mẫu biểu như giá trị Variant chẳng hạn. Có một thói quen trong Visual Basic làm cho Visual Basic nghĩ điều khiển này là chuỗi khi bạn muốn cộng hai giá trị của điều khiển với nhau. Hàm Val() sẽ bảo đảm rằng Visual Basic xem giá trị của điều khiển như con số khi bạn cần trong biểu thức.

Hình 7.2 minh họa mẫu biểu đơn giản cùng với hai hộp nhập và một nút lệnh. Ngay khi người dùng nhập tiền lương năm cuối vào hộp nhập txtLast và tiền lương năm hiện tại vào hộp nhập txtThis, nút lệnh sẽ kích hoạt thủ tục biến cố Click thực hiện phép tính sau:

```
lblAvg = (txtLast.Text + txtThis.Text) / 2
```

Hình 7.2. *Tính giá trị lương trung bình cho hai năm.*

Dù thực tế mọi điều là đúng với biểu thức này, song Visual Basic sẽ không lưu giá trị trung bình của hai hộp nhập vào lblAvg label. Visual Basic hiểu sai rằng bạn muốn nối hai giá trị thay vì cộng lại với nhau.

Ghi chú

Vấn đề có thể phát sinh khi ngôn ngữ sử dụng cùng toán tử + cho hai mục đích khác nhau!

Thay vì sử dụng biểu thức tương đối đơn giản này, bạn phải thay đổi đôi chút hàm Val(). Từ Val() trông giống tên thủ tục biến cố vì cặp dấu ngoặc đơn. Tuy nhiên, Visual Basic có nhiều thủ tục con cài đặt sẵn gọi là hàm. *Tên hàm* kết thúc bằng cặp dấu ngoặc đơn. Bạn sẽ học tất cả các hàm Visual Basic trong Chương 7. Còn bây giờ, chỉ dùng hàm – điều đó quả là dễ – và không nên lo lắng về quy cách chúng làm việc.

Đây là cách sử dụng hàm Val(): đặt chuỗi vào bên trong cặp dấu ngoặc đơn. Giá trị trong cặp dấu ngoặc đơn có thể được biên dịch như một số, ngay cả khi giá trị được lưu dưới dạng chuỗi, Visual Basic sẽ đổi giá trị thành một số tạm để có thể tính kết quả. Lệnh gán trước đòi hỏi bạn áp dụng hàm Val() hai lần. Ví dụ, thay vì

```
lblAvg = (txtLast.Text + txtThis.Text) / 2
```

Bạn phải viết mã lệnh

```
lblAvg = (Val(txtLast.Text) + Val(txtThis.Text)) / 2
```

Val() đảm bảo rằng Visual Basic xem những hộp nhập như các số chứ không phải là chuỗi.

Bằng cách đó, nếu người dùng không nhập số vào cả hai hộp nhập, Visual Basic sẽ sử dụng số 0 cho số được chuyển đổi, giả như không có ký số nào xuất hiện trong các hộp nhập. Còn như một số bất kỳ hoặc tổ hợp ký số hiển thị trong hộp nhập, hàm Val() sẽ dùng những ký số này cho số được chuyển đổi, thường đưa ra các kết quả ngớ ngẩn. Ví dụ, nếu txtLast.Text chứa nội dung 18 years old (lý do người dùng nhập số đó là do hiểu nhầm), hàm Val() sẽ nhận số 18 và sử dụng giá trị 18 trong phép tính trung bình.

Ghi chú

Bạn có thể bắt gặp chương trình tính trung bình lương đơn giản này với cái tên SALAVG.VBP. Chương trình này rất đơn giản. Nó không định dạng phép tính trung bình theo đơn vị tiền tệ đô–la, cho nên có thể có nhiều hơn hoặc ít hơn hai chữ số thập phân. Tuy nhiên, bạn có thể tải xuống và tìm hiểu thủ tục cmdAvg_Click() của chương trình để làm quen với phép tính đang được thực hiện và cách thức hàm Val() trợ giúp quá trình tính toán.

Củng cố

Khi tính toán bằng cách áp dụng dữ liệu của mẫu biểu, bạn có thể chuyển đổi tạm thời những giá trị của điều khiển thành số bằng cách áp dụng hàm Val() trước khi dùng những giá trị này trong biểu thức số. Điều khiển duy trì kiểu dữ liệu Variant ngoại trừ vị trí nơi hàm Val() biến đổi kiểu dữ liệu nhãn thành số.

Bài tập

Kiến thức tổng quát

1. Biến là gì?
2. Câu lệnh nào định nghĩa các biến?
3. Kiểu dữ liệu là gì?
4. Kể tên bảy kiểu dữ liệu trong Visual Basic.
5. Kiểu dữ liệu nào lưu giữ vùng giá trị nhỏ nhất?
6. Kiểu dữ liệu nào lưu giữ vùng giá trị lớn nhất?
7. Các kiểu dữ liệu nào luôn có dấu thập phân?
8. Nghĩa của E là gì khi nó xuất hiện trong các số?
9. Ký hiệu khoa học là gì?
10. Đúng hay Sai: Hằng có thể thay đổi qua chương trình.
11. Đúng hay Sai: Biến có thể khả dụng trong suốt chương trình.
12. Tại sao lại có ích khi định nghĩa tất cả các biến trước khi sử dụng chúng?
13. Điểm khác biệt giữa chuỗi có độ dài thay đổi và chuỗi có độ dài cố định là gì?
14. Đúng hay Sai: Bạn có thể lưu các giá trị thuộc tính điều khiển trong các biến.
15. Đúng hay Sai: Bạn có thể lưu biến trong các giá trị thuộc tính điều khiển.

16. Hãy cho biết thứ tự ưu tiên trong Visual Basic, giá trị nào được gán cho biến bên trái trong từng câu lệnh gán sau?

    A. ResultA = 1 + 2 * 3

    B. ResultB = (1 + 2) * 3

    C. ResultC = 2 ^ 4

    D. Average = 10 + 20 + 30 / 3

    E. Num = 10 – 3 * 2 + 4 / 2

    F. Ans = 10 * 2 ^ 2

17. Toán tử là gì?

18. Điểm khác biệt giữa toán tử + và & là gì?

Lập trình...

19. Định nghĩa biến lưu giữ diện tích chính xác nhà bạn. Sử dụng biến có thể chấp nhận dấu thập phân nhưng không tăng quá mức và định nghĩa biến lớn hơn kích thước bạn thật sự cần.

20. Viết lại ngắn gọn những câu lệnh gán sau:

```
Let SalesPrice = Price / Discount Let Tax = TaxRate * SalesPrice
```

Tìm lỗi kỹ thuật

21. Có gì sai đối với các định nghĩa biến sau:

```
Dim    MyAge     As    Integer
Dim    Height    As    Single
Dim    Weight    As    Double
Dim    Name      As    String
```

22. An Huy đang gặp rắc rối với chương trình Visual Basic của anh ấy. Anh ấy đã khai báo ba biến mà anh ấy muốn sử dụng cho ba tổng số:

```
Dim    DivTotal    As    Single
Dim    DivTotal    As    Single
Dim    DivTotal    As    Single
```

Mục đích của An Huy là lưu những tổng giá trị này cho mỗi phần chia của công ty trong ba biến. Visual Basic không hài

lòng về những gì anh ấy đang thực hiện. Hãy giúp An Huy gỡ rối bởi vì giám đốc đang khó chịu với anh ấy.

23. An Huy nhận ra một lỗi khi gán biến currency sau giá trị. Hãy chỉ cho An Huy biết anh ấy đã làm sai điều gì.

```
Salary = $47,533.90
```

Phần nâng cao

Thêm nhãn thứ ba vào chương trình tính mức lương trung bình SALAVG.VBP được minh họa trong Hình 7.2 sao cho chương trình tính mức trung bình của ba giá trị lương.

Bài 8

So sánh dữ liệu

- Các toán tử quan hệ
- Câu lệnh If
- Xử lý điều kiện False
- Toán tử logic
- Nhiều chọn lựa với Select Case
- Hai dạng Select Case bổ sung

Máy tính không tự suy nghĩ nhưng với sự giúp đỡ của bạn chúng có thể đưa ra quyết định dựa trên giá trị chứa trong điều khiển và biến. Khả năng đưa ra quyết định của Visual Basic cho phép tính lương dựa trên những điều kiện xác định, in báo cáo và kiểm tra hỏi đáp của người dùng v.v.

Bạn chỉ mới học một vài lệnh:

- Lệnh End, dừng chương trình
- Lệnh Dim, định nghĩa biến chương trình
- Lệnh gán, lưu giá trị dữ liệu vào điều khiển và biến

Thêm vào đó, bạn còn hiểu biết đôi chút về các toán tử toán học cơ bản. Trong bài này, bạn sẽ học một số lệnh và toán tử lập trình mới. Bạn có thể dùng kèm với lệnh đã biết để viết chương trình đưa ra quyết định dựa trên dữ liệu.

CÁC TOÁN TỬ QUAN HỆ

Khái niệm

Visual Basic hỗ trợ 6 toán tử vốn trả lại kết quả là True hoặc False dựa trên giá trị dữ liệu. Một khi đã hiểu thấu đáo về những toán tử quan hệ, bạn có thể phối hợp chúng với câu lệnh If nhằm bổ sung tính hoàn chỉnh cho chương trình.

Khái niệm mới

Toán tử quan hệ (Relational operators) so sánh giá trị dữ liệu này với giá trị dữ liệu khác.

Bảng 8.1 mô tả sáu toán tử quan hệ do Visual Basic hỗ trợ. Bạn dùng toán tử quan hệ để so sánh giá trị dữ liệu. Chúng rất dễ sử dụng. Nếu bạn lấy hai số bất kỳ, thì số này sẽ luôn lớn hơn, bằng hoặc nhỏ hơn số kia.

Ghi chú

Các toán tử toán học mà bạn đã làm quen ở bài trước trả về kết quả số. Toán tử quan hệ chỉ trả lại kết quả đúng (True) hay sai (False). Nói cách khác, giá trị dữ liệu lớn hơn giá trị khác thì kết quả là True, hoặc giả giá trị dữ liệu nhỏ hơn giá trị khác thì kết quả sẽ là False.

Bảng 8.1. *Các toán tử quan hệ*

Toán tử	Cách dùng	Mô tả
>	Sales > Goal	Toán tử lớn hơn. Trả về giá trị True nếu giá trị bên trái dấu > lớn hơn về mặt số lượng hoặc thứ tự abc so với giá trị bên phải. Ngược lại là False.
<	Pay < 2000.00	Toán tử nhỏ hơn. Trả về giá trị True nếu giá trị bên trái dấu < nhỏ hơn về mặt số lượng hoặc thứ tự abc so với giá trị bên phải. Ngược lại là False.
=	Age = Limit	Toán tử bằng. Trả về giá trị True trong trường hợp giá trị ở hai bên dấu = bằng nhau. Ngược lại là False.
>=	FirstName >= "Mike"	Toán tử lớn hơn hoặc bằng. Trả về giá trị True trong trường hợp giá trị bên trái dấu >= lớn hơn hoặc bằng về mặt số lượng hoặc thứ tự abc so với giá trị bên phải. Ngược lại là False.

<=	Num <= lblAmt.Caption	Toán tử nhỏ hơn hoặc bằng. Trả về giá trị True khi giá trị bên trái dấu <= nhỏ hơn hoặc bằng về mặt số lượng hoặc thứ tự abc so với giá trị bên phải. Ngược lại là False.
<>	txtAns.Text <> "Yes"	Toán tử không bằng. Trả về giá trị True khi giá trị ở bên trái dấu <> khác về mặt số lượng hoặc thứ tự abc so với giá trị bên phải. Ngược lại là False.

Khái niệm mới

Bảng mã* ASCII là danh sách ký tự *trình bày các con số tương ứng.

Mọi toán tử quan hệ làm việc trên cả giá trị bảng chữ cái lẫn số. Bạn có thể so sánh một loại số bất kỳ với một số khác, hay chuỗi này với chuỗi khác. Khi bạn so sánh chuỗi, Visual Basic dùng bảng ASCII để xác định cách so sánh các ký tự. Ví dụ bảng ASCII cho biết ký tự *A có giá trị số ASCII bằng* 65 nhỏ hơn ký tự *B có giá trị số ASCII bằng 66*. Lưu ý tất cả ký tự hoa đều nhỏ hơn ký tự thường. Vì vậy, chữ *ST nhỏ hơn St*.

Muốn nắm bắt cách thức toán tử quan hệ thực hiện, bạn phải hiểu cách sử dụng kết quả True hoặc False của chúng. Lệnh If được giới thiệu trong mục kế tiếp sẽ giải thích quy cách bạn có thể áp dụng kết quả True và False để tạo quyết định trong chương trình. Trước khi đọc mục tiếp theo, bạn phải hiểu cách toán tử này so sánh giá trị. Để tự kiểm tra nhanh, bảo đảm bạn phải hiểu cột *Kết quả* trong Bảng 8.2 trước khi tìm hiểu kỹ lưỡng hơn.

Bảng 8.2. *Các kết quả quan hệ*

Quan hệ	Kết quả
10 > 5	True
5 > 10	False
5 < 10	True

"Apple" <= "Orange"	True
"Mac Donald" < "Mc Donald"	True
0 >= 0	True
0 <= 0	True
1 <> 2	True
2 >= 3	False

Hãy giữ mỗi bên một kiểu dữ liệu thích hợp: cả hai bên của biểu thức trong toán tử quan hệ phải có cùng kiểu dữ liệu hoặc ít nhất có kiểu dữ liệu tương thích. Nói cách khác, bạn không thể so sánh chuỗi với kiểu dữ liệu số. Nếu cố thực hiện, bạn sẽ nhận được lỗi không hợp kiểu (Type mismatch) bởi lẽ kiểu dữ liệu đó không phù hợp.

Có thể so sánh kiểu dữ liệu số này với kiểu dữ liệu số khác. Nói cách khác, bạn có thể kiểm tra liệu giá trị chính xác đơn nhỏ hơn hay lớn hơn giá trị số nguyên. Tuy nhiên, hãy cẩn thận khi so sánh bằng với một số không phải là số nguyên. Số chính xác thì khó trình bày bên trong. Ví dụ, nếu bạn đã gán 8.3221 cho biến có độ chính xác đơn và gán 8.3221 cho biến có độ chính xác đơn khác, Visual Basic sẽ trả lại giá trị False, giả như bạn so sánh bằng hai giá trị này. Trong bộ nhớ, biến có thể lưu giữ số thực sự là 8.322100001 vì lỗi làm tròn số xảy ra với số thập phân vô nghĩa thông thường. Tuy nhiên, bạn có thể đảm bảo việc so sánh bằng hai giá trị Currency, bởi lẽ Visual Basic duy trì sự chính xác với hai chữ số thập phân.

Ghi chú

Toán tử quan hệ đôi khi được gọi là toán tử điều kiện (conditional operator) bởi vì chúng kiểm tra điều kiện là True hoặc False.

Củng cố

Toán tử quan hệ so sánh giá trị này với giá trị khác. Bạn có thể so sánh bằng, không bằng và lớn hơn. Các toán tử quan hệ xử lý cả dữ liệu chuỗi lẫn dữ liệu số. Bản thân toán tử quan hệ sẽ không có giá trị lắm. Tuy nhiên, bạn có thể dùng chúng hầu so sánh dữ liệu bằng cách dùng lệnh If, mà bạn sẽ học trong mục kế tiếp.

CÂU LỆNH If

Khái niệm

Lệnh If sử dụng toán tử quan hệ để kiểm tra giá trị. Lệnh này thực hiện một trong hai khả năng hành động của mã lệnh, phụ thuộc vào kết quả kiểm tra. Trong bài trước, bạn đã tìm hiểu cách thức Visual Basic thi hành lệnh Dim và lệnh gán theo thứ tự bạn nhập chúng trong chương trình. Với câu lệnh If, Visual Basic kiểm tra xem có thi hành khối mã lệnh không. Nói cách khác, lệnh If dùng toán tử quan hệ nhằm kiểm tra dữ liệu và có thể thi hành một hoặc nhiều dòng lệnh theo sau, còn tùy thuộc vào kết quả kiểm tra.

Lệnh If thực hiện quyết định. Nếu một quan hệ kiểm tra là True, phần chính của câu lệnh If sẽ thi hành. Sau đây là dạng thức của lệnh If:

```
If relationalTest Then
    ' One or more Visual Basic statements
End If
```

Lệnh End If báo cho Visual Basic biết vị trí phần nội dung của câu lệnh If sẽ kết thúc. Giả sử người dùng nhập một mức lương vào điều khiển hộp nhập txtSales. Câu lệnh If sau sẽ tính số tiền thưởng dựa trên mức lương đó:

```
If (txtSales.Text > 5000.00) Then
    Bonus = Val(txtSales.Text) * .12
End If
```

Nhớ rằng Visual Basic lưu số 0 trong mọi biến mà bạn không khởi tạo trước đó. Vì vậy, Bonus có giá trị bằng 0 trước lúc lệnh If thi hành. Lệnh If thi hành, mã lệnh sẽ thay đổi biến Bonus chỉ khi giá trị của thuộc tính txtSales.Text lớn hơn 5000.00. Hàm Val() biến đổi dữ liệu Variant của hộp nhập thành giá trị số để tính toán. Bằng cách này, lệnh If có thể đọc như sau: *Nếu* mức lương *lớn hơn $5.000.00 thì tiền thưởng dựa trên giá trị của mức lương.*

Phần nội dung của lệnh If có thể có nhiều lệnh. Lệnh If dưới đây sẽ tính tiền thưởng, lương phải trả, và phụ cấp dựa trên giá trị nhập vào txtSales:

```
If (txtSales.Text > 5000.00) Then
    Bonus = Val(txtSales.Text) * .12
    CostOfSales = Val(txtSales.Text) * .41
    ReorderCost = Val(txtSales.Text) * .24
End If
```

Ba câu lệnh trong thân lệnh If chỉ thi hành trong trường hợp điều kiện txtSales.Text > 5000.00 là đúng. Giả sử mã lệnh này chứa câu lệnh gán khác ngay sau lệnh End If. Câu lệnh gán đó ở ngoài thân lệnh If, cho nên kết quả True hoặc False của điều kiện chỉ ảnh hưởng đến thân lệnh If. Vì vậy, quá trình tính thuế trong thủ tục sau sẽ thực hiện bất chấp mức lương lớn hơn hay nhỏ hơn $5.000.00:

```
If (txtSales.Text > 5000.00) Then
    Bonus = Val(txtSales.Text) * .12
    CostOfSales = Val(txtSales.Text) * .41
    ReorderCost = Val(txtSales.Text) * .24
End If
Tax = .12 *
Val(txtSales.Text)
```

Mách nước

Cặp dấu ngoặc đơn không bắt buộc phải bao quanh biểu thức kiểm tra quan hệ nhưng chúng giúp phân biệt phần kiểm tra với phần còn lại của mã lệnh.

Bạn có hiểu cách chương trình thực hiện quyết định bằng cách sử dụng If chưa? Thân lệnh If chỉ thi hành nếu kiểm tra quan hệ là True. Ngược lại, phần còn lại của chương trình sẽ tiếp tục như thường lệ.

Có dạng rút gọn của If mà bạn có thể không chú ý. Câu lệnh If trên một dòng thuộc dạng thức sau:

```
If relationalTest Then VBStatement
```

Lệnh If một dòng không đòi hỏi lệnh End If bởi vì kiểm tra quan hệ và phần thân lệnh If tập trung trên cùng dòng. Lệnh If một dòng gây khó dễ cho quá trình bảo trì chương trình. Giả như bạn muốn thêm lệnh vào thân lệnh If, bạn phải chuyển lệnh If một dòng thành câu lệnh If nhiều dòng và có lẽ bạn sẽ quên không bổ sung câu lệnh End If. Chính vì thế, ngay cả khi phần thân lệnh If chỉ có một lệnh, hãy viết lệnh này như câu lệnh If nhiều dòng If–End If.

Củng cố

Lệnh If xác định xem mã lệnh có thi hành không. If sẽ kiểm tra điều kiện True hoặc False của biểu thức kiểm tra quan hệ. Nếu dữ liệu kiểm tra quan hệ là True, Visual Basic sẽ thi hành thân lệnh If. Còn như dữ liệu kiểm tra quan hệ là False, Visual Basic sẽ chuyển qua phần thân của câu lệnh. Không có điều gì xảy ra, mã lệnh theo sau lệnh End If sẽ thi hành như thường lệ.

XỬ LÝ ĐIỀU KIỆN False

Khái niệm

Ngược lại với If sẽ thi hành mã lệnh dựa trên điều kiện True của cơ chế kiểm tra quan hệ, lệnh Else sẽ thi hành mã lệnh dựa trên điều kiện False của kiểm tra quan hệ. Else là một phần thực sự của lệnh If. Phần này sẽ giải thích đầy đủ về câu lệnh If–Else. Nó chỉ cho bạn cách có thể thi hành phần mã lệnh này hoặc phần mã lệnh khác, phụ thuộc vào cơ chế kiểm tra quan hệ.

Lệnh Else, phần mở rộng của lệnh If, xác định mã lệnh thi hành nếu biểu thức kiểm tra quan hệ là False. Sau đây là dạng thức đầy đủ của câu lệnh If với Else:

```
If relationalTest Then
' One or more Visual Basic statements
Else
' One or more Visual Basic statements
End If
```

Lập trình viên thường gọi câu lệnh If đầy đủ này là lệnh If–Else. Lệnh If–Else đôi khi gọi là lệnh *loại trừ lẫn nhau* (mutually exclusive). Cụm từ "loại trừ lẫn nhau" *đơn giản có nghĩa* tập hợp mã lệnh này hay mã lệnh khác sẽ thi hành, nhưng không thể là cả hai. Lệnh If–Else có chứa hai tập hợp mã lệnh – đó là hai phần thân chứa một hoặc nhiều câu lệnh Visual Basic – và chỉ một tập hợp sẽ thi hành, phụ thuộc vào kết quả của If. Lệnh If có thể đúng (True) hoặc sai (False). Vì vậy, hoặc phần thứ nhất hoặc phần thứ hai của mã lệnh trong câu lệnh If–Else sẽ thi hành.

Tóm tắt

Giả sử nhân viên bán hàng sẽ nhận được một khoản tiền thưởng nếu đạt doanh số cao (trên $5.000.00) hoặc phải chịu phạt nếu doanh số thấp (dưới $5.000.00). Lệnh If–Else trong Ví dụ 8.1 chứa mã lệnh cần thiết để thưởng hay phạt nhân viên bán hàng. Thân mã lệnh If tính tiền thưởng đã thực hiện ở phần trước. Thân mã lệnh Else sẽ trừ $25 vào lương của nhân viên bán hàng, lưu trong biến PayAmt, giả như mã số lương không có.

Ví dụ 8.1 gồm có một *phân đoạn mã lệnh* (code fragment) - đôi khi được gọi là code snippet – bởi vì ví dụ không chỉ ra các định nghĩa biến hay mã lệnh trước đó đã khởi tạo biến PayAmt chứa lương của nhân viên bán hàng. Ví dụ 8.1 chỉ gồm mã lệnh If–Else cần để thưởng hay phạt quá trình nỗ lực của nhân viên bán hàng.

Củng cố

Lệnh If xử lý kết quả True của biểu thức kiểm tra điều kiện và Else xử lý kết quả False. Bằng cách sử dụng lệnh If–Else, bạn có thể đẩy mạnh khả năng kiểm tra dữ liệu của Visual Basic và khả năng thực hiện quyết định.

Ví dụ 8.1. *Xác định mức thưởng phạt.*

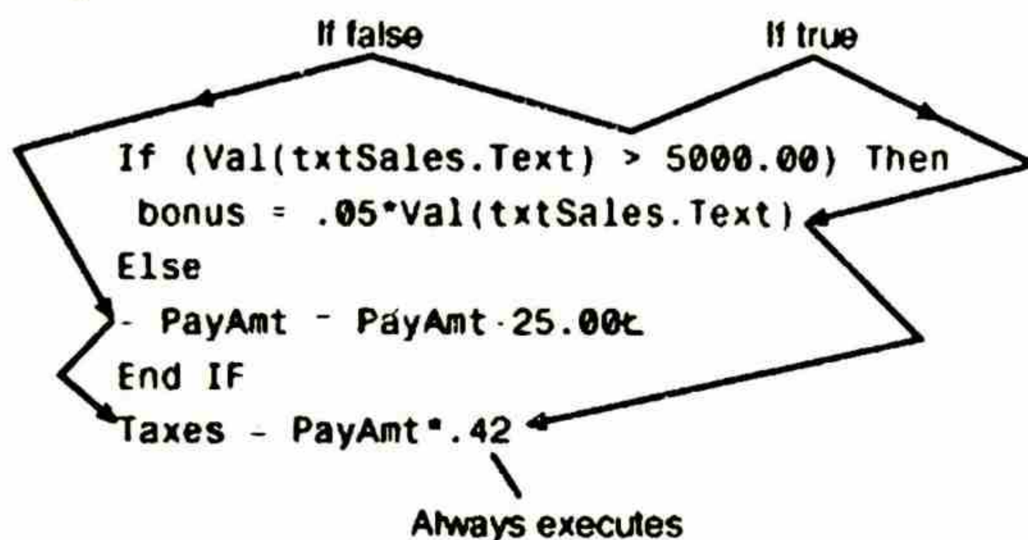
```
1: If (Val(txtSales.Text) > 5000.00) Then
2: Bonus = .05 * Val(txtSales.Text)
3: Else
4: PayAmt = PayAmt – 25.00
5: End If
6: Taxes = PayAmt * .42
```

Phân tích

Lưu ý dòng 6 sẽ tính số tiền thuế. Không cần đến tác động của biểu thức kiểm tra quan hệ của lệnh If–Else, dòng 6 luôn thi hành. Dòng 6 không thuộc phần thân của lệnh If lẫn lệnh Else.

Dòng 1 kiểm tra liệu giá trị hàng hóa của nhân viên bán hàng được lưu trong hộp nhập txtSales lớn hơn $5,000.00. Nếu biểu thức kiểm tra quan hệ là True, dòng 2 sẽ tính mức thưởng. Còn như False, lệnh Else sẽ thi hành (dòng 4).

Như Hình 8.1 minh họa, mã lệnh thuộc phần thân lệnh If sẽ thực hiện, hoặc mã lệnh thuộc phần thân lệnh Else sẽ thực thi, song không bao giờ thực hiện cả hai. Visual Basic quyết định trong dòng 1 phép gán nào sẽ thực hiện.



Hình 8.1. *Thân lệnh If hoặc thân lệnh Else sẽ thi hành, nhưng không thể là cả hai.*

Dòng 4 giới thiệu lệnh gán, có lẽ lúc đầu làm bạn ngạc nhiên. Dòng này xuất hiện để thực hiện lệnh tính tiền lương bằng tiền lương sẽ trả trừ đi 25. Bạn biết rằng không có cái gì có thể bằng chính nó trừ đi 25. Trong toán học, dấu bằng hoạt động như một đối trọng hai vế dấu bằng. Tuy nhiên, trong Visual Basic, khi dấu bằng không được sử dụng trong biểu thức kiểm tra quan hệ của If, nó là phép gán vốn tính toán mọi thứ bên phải dấu bằng (=) và lưu giá trị đó vào trong biến bên trái dấu (=). Dòng 4 sẽ lấy PayAmt trừ đi 25, sau đó gán kết quả trở lại PayAmt. Cuối cùng, giá trị của nó nhỏ hơn giá trị của PayAmt là 25.

Ghi chú

Khi biến xuất hiện ở hai bên dấu = trong phép gán, biến sẽ được cập nhật theo vài cách.

TOÁN TỬ *Logic*

Khái niệm

Ba toán tử bổ sung And, Or, và Not, trông giống lệnh hơn là toán tử. And, Or, và Not được gọi là toán tử logic. Toán tử này cho phép bạn tăng khả năng của các toán tử quan hệ bằng cách mở rộng quá trình kiểm tra trong câu lệnh If. Chúng cho phép bạn kết hợp hai hay nhiều biểu thức kiểm tra quan hệ.

Bảng 8.3 mô tả các toán tử logic, thực hiện y như nội dung mô tả về chúng.

Bảng 8.3. *Toán tử logic*

Toán tử	Cách dùng	Mô tả
And	If (A > B) And (C < D)	Trả lại giá trị True nếu cả hai bên toán tử And là True. Vì vậy, A phải lớn hơn B và C nhỏ hơn D. Ngược lại biểu thức trả lại giá trị False.
Or	If (A > B) Or (C < D)	Trả lại giá trị True nếu bên này, hoặc bên kia của toán tử Or là True. Do đó, A phải lớn hơn B hoặc C phải nhỏ hơn D. Giả như cả hai bên toán tử Or đều False, toàn bộ biểu thức sẽ trả lại kết quả False.
Not	If Not(Ans = "Yes")	Cho kết quả ngược lại với kết quả True hoặc False. Vì vậy, trường hợp Ans chứa "Yes", toán tử Not sẽ đảo kết quả True thành False.

Qua Bảng 8.3 ta thấy, toán tử logic And và Or cho phép bạn phối hợp nhiều biểu thức kiểm tra quan hệ trong câu lệnh If. Toán tử Not phủ định kết quả của biểu thức kiểm tra quan hệ. Bạn có thể đảo ngược hoàn toàn bằng điều kiện Not. Not có thể tạo biểu thức kiểm tra quan hệ phức tạp, và bạn nên sử dụng nó thận trọng. Chẳng hạn, lệnh If cuối cùng trong Bảng 8.3 có thể dễ dàng chuyển thành If (Ans <> "Yes") nhằm loại bỏ điều kiện Not.

Mã lệnh thường phải thực hiện phép gán, in thông báo, hoặc hiển thị nhãn nếu hai hay nhiều điều kiện là True. Toán tử logic làm cho điều kiện dễ dàng phối hợp với mã lệnh. Giả sử bạn muốn thưởng nhân viên nếu tổng số tiền bán hàng lớn hơn $5.000 và nếu anh ta bán nhiều hơn 10.000 đơn vị sản phẩm khác nhau. Không có toán tử And, bạn phải nhúng câu lệnh If trong thân câu lệnh If khác. Ví dụ:

```
If (Sales > 5000.00) Then
    If (UnitsSold > 10000) Then
    Bonus = 50.00
    End If
End If
```

Dưới đây là mã lệnh tương tự được viết dưới dạng lệnh If đơn lẻ. Sau này đặt trường hợp bạn cần cập nhật chương trình, cũng dễ thay đổi thôi.

```
If (Sales > 5000.00) And (UnitsSold > 10000) Then
    Bonus = 50.00
End If
```

Bạn có thể viết lại mã lệnh này như thế nào bằng lệnh If để tính tiền thưởng, giả như nhân viên bán hoặc hơn $5.000 hoặc nhiều hơn 10.000 đơn vị sản phẩm? Sau đây là mã lệnh:

```
If (Sales > 5000.00) Or (UnitsSold > 10000) Then
    Bonus = 50.00
End If
```

Tóm tắt

Ví dụ 8.2 có chứa lệnh If–Else vốn kiểm tra dữ liệu hai nhóm trong một công ty và tính toán giá trị từ dữ liệu.

Củng cố

Toán tử logic cho phép bạn phối hợp hai hay nhiều điều kiện kiểm tra. Không có các toán tử logic, bạn phải viết mã lệnh dài hơn bằng những lệnh If lồng vào nhau.

Ví dụ 8.2. *Tính lương các nhóm trong một công ty.*

```
If (DivNum = 3) Or (DivNum = 4) Then
DivTotal = DivSales3 + DivSales4
GrandDivCosts = (DivCost3 * 1.2) + (DivCost4 * 1.4)
Else
DovTotal = DivSales1 + DivSales2
GrandDivCosts = (DivCost1 * 1.1) + (DivCost5 * 1.9)
End If
```

Phân tích

Giả sử người dùng mã lệnh trong Ví dụ 8.2 làm chủ công ty gồm có 4 nhóm. Các nhóm east coast được đánh số là 3 và 4, các nhóm west coast được đánh số là 1 và 2. Ví dụ 8.2 sẽ tính tổng lương và tổng chi

phí cho các nhóm theo coast, còn tùy thuộc vào giá trị DivNum. Bạn phải giả sử tất cả các biến đã được định nghĩa và khởi tạo với những giá trị đúng.

Nếu DivNum chứa số 3 hoặc 4, người dùng đang yêu cầu tính cho east coast, mã lệnh trong dòng 2 và 3 sẽ thi hành để đưa ra cặp giá trị cho east coast. Trường hợp DivNum không chứa giá trị 3 hoặc 4, chương trình giả thiết DivNum chứa 1 hoặc 2, và cặp giá trị west coast được tính từ dòng 5 đến dòng 6.

NHIỀU CHỌN LỰA VỚI *Select Case*

Khái niệm

Lệnh If so sánh dữ liệu có hiệu quả trong trường hợp một hoặc hai biểu thức kiểm tra quan hệ được tạo. Tuy nhiên, khi bạn phải kiểm tra nhiều hơn hai điều kiện, lệnh If trở nên khó bảo trì. Toán tử logic trợ giúp loại điều kiện nhất định nào đó. Đôi khi bạn buộc phải lồng nhiều câu lệnh If–Else trong câu lệnh If–Else khác.

Hãy xem xét lệnh If minh họa ở Ví dụ 8.3. Mặc dù toán tử logic của lệnh If đơn giản, song mã lệnh sau rất khó theo dõi.

Ví dụ 8.3. *Các câu lệnh If–Else lồng vào nhau làm cho công việc trở nên phức tạp.*

```
If (Age = 5) Then
    lblTitle.Text = "Kindergarten"
Else
    If (Age = 6) Then
    lblTitle.Text = "1st Grade"
    Else
    If (Age 7) Then
    lblTitle.Text = "2nd Grade"
    Else
If (Age = 8) Then
    lblTitle.Text = "3rd Grade"
    Else
    If (Age = 9) Then
```

```
    lblTitle.Text = "4th Grade"
    Else
    If (Age = 10) Then
    lblTitle.Text = "5th Grade"
    Else
    If (Age = 11) Then
    lblTitle.Text = "6th
Grade"
    Else
    lblTitle.Text = "Advanced"
    End If
    End If
    End If
    End If
    End If
    End If
End If
```

Visual Basic hỗ trợ một câu lệnh, gọi là Select Case, vốn xử lý các điều kiện có nhiều lựa chọn tốt hơn If–Else. Sau đây là dạng thức của lệnh Select Case:

```
Select Case Expression
Case value
    ' One or more Visual Basic statements
Case value
    ' One or more Visual Basic statements
    [Case value
' One or more Visual Basic statements ]
[Case Else:
' One or more Visual Basic statements]
End Select
```

Dạng thức của Select Case làm cho lệnh này giống y hệt lệnh If–Else lồng nhau phức tạp, nhưng bạn sẽ thấy câu lệnh Select Case thật sự dễ viết chương trình và bảo trì hơn so với bản sao If–Else của chúng.

Expression có thể là biểu thức Visual Basic bất kỳ như phép tính, giá trị chuỗi hoặc giá trị số chẳng hạn – sẽ có kết quả là giá trị số nguyên hoặc giá trị chuỗi. *Values* phải là các giá trị số nguyên hoặc giá trị chuỗi so khớp với kiểu dữ liệu của *Expression*.

Lệnh Select Case có ích khi bạn phải thực hiện nhiều lựa chọn dựa trên giá trị dữ liệu. Select Case có thể có hai hay nhiều phần giá trị Case. Mã lệnh thi hành phụ thuộc vào giá trị nào phù hợp với *Expression*. Nếu không có giá trị nào phù hợp với *Expression*, mã lệnh trong phần Case Else sẽ thi hành giống như bạn viết mã lệnh cho Case Else vậy. Ngược lại, không có điều gì xảy ra thì tiếp tục điều khiển lệnh đứng sau End Select.

Lưu ý

Không nên sử dụng Select Case khi lệnh If hoặc lệnh If–Else đơn giản đáp ứng được. Cơ chế kiểm tra logic của lệnh If ít phức tạp và thậm chí còn rõ ràng hơn lệnh Select Case. Nếu không cần so sánh nhiều hơn hai giá trị, thì nên phối hợp lệnh If và If–Else vì chúng rất đơn giản.

Tóm tắt

Cách nhanh nhất để học Select Case là xem ví dụ về nó. Ví dụ 8.4 chứa phiên bản Select Case, các phép gán kết quả xếp hạng học tập biểu diễn trong Ví dụ 8.3. Select Case sắp xếp nhiều tùy chọn thành dạng thức dễ quản lý hơn.

Củng cố

Lệnh Select Case là lệnh thay thế khá ăn ý cho các điều kiện If-Else lồng nhau và dài khi một trong nhiều tùy chọn có khả năng. Bạn khởi tạo chương trình Visual Basic để thi hành tập hợp lệnh Visual Basic từ danh sách lệnh bên trong Select Case.

Ví dụ 8.4. *Sử dụng Select Case để đơn giản hóa các lệnh If–Else lồng nhau phức tạp.*

```
1: Select Case Age
2: Case 5: lblTitle.Text = "Kindergarten"
3: Case 6: lblTitle.Text = "1st Grade"
4: Case 7: lblTitle.Text = "2nd Grade"
5: Case 8: lblTitle.Text = "3rd Grade"
6: Case 9: lblTitle.Text = "4th Grade"
```

```
7: Case 10: lblTitle.Text = "5th Grade"
8: Case 11: lblTitle.Text = "6th Grade"
9: Case Else: lblTitle.Text = "Advanced"
10: End Select
```

Phân tích

Nếu biến Age lưu giữ giá trị 5, nhãn "Kindergarten" được gán ở dòng 2. Trường hợp biến Age lưu giữ giá trị 6, nhãn "1st Grade" được gán ở dòng 3. Toán tử logic sẽ tiếp tục gán cho đến dòng 9. Giả như biến Age lưu giữ giá trị ngoài các giá trị từ 5 đến 11, dòng 9 sẽ gán "Advanced" cho nhãn.

Thân của từng tùy chọn Case có thể chứa nhiều lệnh, cũng như thân của lệnh If hoặc If–Else có thể chứa nhiều lệnh. Visual Basic sẽ thi hành tất cả các lệnh bất kỳ của tùy chọn Case tương ứng đến khi bắt gặp tùy chọn kế tiếp. Một khi Visual Basic đã thi hành tùy chọn Case phù hợp, nó sẽ chuyển nhanh qua những lệnh Case còn lại và tiếp tục với mã lệnh đặt ngay sau lệnh End Select.

Ghi chú

Lập trình viên thường kích hoạt quá trình thi hành các thủ tục đầy đủ, như thủ tục biến cố, từ câu lệnh Case. Như bạn đã học ở Bài 8, thay vì đặt nhiều câu lệnh trong thân của lệnh If–Else hoặc lệnh Case, bạn có thể thi hành thủ tục bao gồm tất cả các lệnh vốn sẽ thi hành khi điều kiện được cho là đúng.

HAI DẠNG *Select Case* BỔ SUNG

Khái niệm

Select Case của Visual Basic là một trong những câu lệnh tùy chọn hoàn hảo nhất trong bất cứ ngôn ngữ lập trình nào. Pascal, C, và C++ – mọi ngôn ngữ lập trình thông dụng – đều có những câu lệnh hoạt động giống hệt lệnh Select Case, nhưng Select Case còn cung cấp hai dạng thức khác cho phép sửa đổi cách thức tùy chọn Case tương ứng được thực hiện.

Hai dạng bổ sung chỉ khác đôi chút so với Select Case chuẩn mà bạn đã học ở bài trước. Chúng cho phép mở rộng khả năng của Select Case sao cho Visual Basic có thể tạo tùy chọn Case phù hợp trên cả hai biểu thức kiểm tra quan hệ lẫn trên vùng giá trị. Sau đây là dạng bổ sung đầu tiên:

```
Select Case Expression
Case Is relation:
    ' One or more Visual Basic statement
Case Is relation:
' One or more Visual Basic statement
[Case Is relation:
' One or more Visual Basic statement ]
[Case Else:
' One or more Visual Basic statement]
End Select
```

relation có thể là biểu thức kiểm tra quan hệ bất kỳ nào đó mà bạn muốn thực hiện *Expression* ở đầu lệnh Select Case. Lệnh Select Case chuẩn so sánh giá trị *Expression* với giá trị tùy chọn Case chính xác đã thảo luận ở mục trước. Khi bạn sử dụng tùy chọn Case Is, mỗi tùy chọn Case có thể tương ứng với một biểu thức kiểm tra quan hệ.

Sau đây là dạng thức thứ hai của lệnh Select Case bổ sung:

```
Select Case Expression
Case expr1 To expr2:
' One or more Visual Basic statement
Case expr1 To expr2:
' One or more Visual Basic statement
[Case expr1 To expr2:
' One or more Visual Basic statement ]
[Case Else:
' One or more Visual Basic statement ]
End Select
```

Dòng tùy chọn Case đòi hỏi vùng giá trị, như dòng từ 4 đến 6 chẳng hạn. Tùy chọn To Select Case cho phép bạn so sánh giá trị vùng thay vì giá trị quan hệ hoặc độ chính xác.

Mách nước

Bạn có thể kết hợp các dạng thức mở rộng của Select Case với lệnh Select Case chuẩn để có hai hay nhiều dạng tùy chọn Case sẽ xuất hiện trong cùng một lệnh.

Tóm tắt

Mã lệnh trong Ví dụ 8.4 chứa lỗi logic nhỏ. Mã lệnh xử lý mọi giá trị Age lớn hơn 11, nhưng nó không xử lý giá trị Age nhỏ hơn 5. Vì vậy, nếu Age chứa giá trị 4, Ví dụ 8.4 sẽ gán "Advanced" cho nhãn. Tuy nhiên, giả như bạn thêm biểu thức kiểm tra quan hệ vào tùy chọn Case đầu tiên, như được minh họa trong Ví dụ 8.5, mã lệnh có thể xử lý giá trị Age.

Củng cố

Các tùy chọn Select Case mở rộng khả năng của Select Case kể cả lệnh tùy chọn bất kỳ thuộc ngôn ngữ khác. Sử dụng hai tùy chọn Case đã học trong mục này, bạn có thể viết mã lệnh nhiều tùy chọn trên ba loại thích hợp:

- Tùy chọn Case chính xác với *Expression* của Select Case
- Tùy chọn Case tương ứng với *Expression* của Select Case
- Vùng tùy chọn Case tương ứng với *Expression* của Select Case

Ví dụ 8.5. *Dùng Select Case để giảm bớt tính phức tạp của các câu lệnh If-Else lồng nhau.*

```
1: Select Case Age
2: Case Is <5: lblTitle.Text = "Too young"
3: Case 5: lblTitle.Text = "Kindergarten"
4: Case 6: lblTitle.Text = "1st Grade"
5: Case 7: lblTitle.Text = "2nd Grade"
6: Case 8: lblTitle.Text = "3rd Grade"
7: Case 9: lblTitle.Text = "4th Grade"
8: Case 10: lblTitle.Text = "5th Grade"
9: Case 11: lblTitle.Text = "6th Grade"
10: Case Else: lblTitle.Text = "Advanced"
11: End Select
```

Phân tích

Bằng cách kiểm tra giá trị nhỏ hơn 5 ở dòng 1, Ví dụ 8.5 đảm bảo cả hai độ tuổi trẻ hơn và già hơn đều chứa trong các tùy chọn Case.

Mách nước

Dù cho dòng Case Else là tùy chọn, cũng nên áp dụng nó nếu không biết chính xác vùng giá trị khả dĩ của Expression.

Tóm tắt

Ví dụ 8.6 nêu ra vấn đề Select Case tương tự, dùng tùy chọn To cho một số giá trị Case. Mỗi tùy chọn Case kiểm tra vùng giá trị có sẵn. Những độ tuổi nằm trong nhóm, và tiêu đề thích hợp được cập nhật tùy thuộc vào các nhóm tương ứng.

Ví dụ 8.6. *Sử dụng vùng Select Case để phân loại nhiều nhóm.*

```
1: Select Case Age
2: Case Is <5: lblTitle.Text = "Too young"
3: Case 5: lblTitle.Text = "Kindergarten"
4: Case 6 To 11: lblTitle.Text = "Elementary"
5: lblSchool.Text = "Lincoln"
6: Case 12 To 15: lblTitle.Text = "Intermediate"
7: lblSchool.Text = "Washington"
8: Case 16 To 18: lblTitle.Text = "High School"
9: lblSchool.Text = "Betsy Ross"
10: Case Else: lblTitle.Text = "College"
11: lblSchool.Text = "University"
12: End Select
```

Phân tích

Ví dụ 8.6 bao gồm từng tùy chọn Select Case có sẵn. Trong lệnh Select Case ở dòng 1, giá trị Age được so sánh với nhiều khả năng của nó trong suốt phần còn lại của lệnh Select Case. Nếu Age lưu giữ giá trị nhỏ hơn 5, cập nhật nhãn trong dòng 2 để làm mới tuổi của trẻ, không thi hành điều gì khác. Visual Basic cho rằng tùy chọn Case đầu tiên trong dòng 2 là tùy chọn Case một dòng, và chương trình tiếp tục với dòng lệnh tiếp theo lệnh End Select (dòng 12) ngay khi dòng 2 hoàn thành.

Dòng 3 so sánh biến Age với giá trị 5, và gán chuỗi "Kindergarten" cho nhãn nếu Age bằng 5.

Dòng 4–9 so sánh liệu Age nằm trong khoảng 3 giá trị khác nhau và từ đó cập nhật 2 giá trị nhãn. Trường hợp tất cả lệnh Case không ở ngay dòng 9, tùy chọn Case trong dòng 10 sẽ thực hiện và giả sử Age lưu giữ giá trị lớn hơn 18. (Nếu Age lưu giữ giá trị bất kỳ nhỏ hơn 18, lệnh Case ở trước sẽ giành quyền điều khiển.)

Bài tập

Kiến thức tổng quát

1. Điều kiện là gì?
2. Đúng hay Sai: Biểu thức kiểm tra quan hệ đưa ra một trong ba kết quả.
3. Đúng hay Sai: Visual Basic yêu cầu cung cấp dấu ngoặc đơn bao quanh toán tử kiểm tra quan hệ.
4. Sáu toán tử quan hệ là gì?
5. Hãy xác định kết quả Đúng hay Sai cho các biểu thức kiểm tra quan hệ sau:

   A. "zz" < "ZZ"

   B. 161 <= 161

   C. 3.4 < 4

   D. (3 <> 4) And (5 = 5)

   E. (3 <> 4) Or (5 = 5)

   F. (3 = 3) And (10 < 9)
6. Tên bảng xác định cách thức giá trị chuỗi được so sánh là gì?
7. Lệnh nào trong Visual Basic thực hiện quyết định?
8. Lệnh nào trong Visual Basic quy định tiến trình hành động đúng hoặc sai?
9. Đúng hay Sai: Thân của lệnh If có thể chứa nhiều lệnh, nhưng thân của lệnh Else phải chứa tối đa một lệnh.
10. Lệnh If lồng nhau là gì?
11. Lệnh nào giúp bỏ qua câu lệnh If–Else chán ngắt và rối rắm?
12. Có bao nhiêu loại tùy chọn Case?
13. Điều gì sẽ xảy đến với từng tùy chọn Case không đúng và không có tùy chọn Case Else dành cho lệnh Select Case riêng biệt?
14. Điều gì sẽ xảy ra nếu từng tùy chọn Case không đúng và có một tùy chọn Case Else trên câu lệnh Select Case?
15. Tùy chọn Case nào sẽ kiểm tra vùng giá trị?
16. Tùy chọn Case sẽ kiểm tra giá trị quan hệ?

Viết mã lệnh...

17. Viết lại lệnh If lồng nhau dưới đây bằng toán tử logic:

```
If (M = 3) Then
If (P = 4) Then
TestIt = "Yes"
End If
End If
```

18. Viết lại lệnh If sau đây bỏ qua toán tử Not để làm rõ mã lệnh:

```
If Not(d < 3) Or Not(p >= 9) Then
```

19. Viết lại Ví dụ 8.6 nhằm thông báo "Age Error" sẽ xuất hiện trong thuộc tính lblTitle.Text, giả như Age chứa giá trị nhỏ hơn 0.

Tìm lỗi kỹ thuật

20. Lệnh If sau đây có gì sai?

```
If (A < 1) And (C >= 8) Then
lblName.Text = "Overdrawn"
Else
lblName.Text = "Underdrawn"
End Else
End If
```

Phần nâng cao

Hãy làm cho lệnh If và lệnh Select Case tương ứng sau đây trở nên thích hợp và dễ bảo trì hơn? Nhớ rằng bạn muốn duy trì mọi thứ đơn giản và rõ ràng.

```
If (grade >= 70) Then
lblLetter.Text = "Passing"
Else
lblLetter.Text = "Failing"
End If
```

Hay

```
Select Case grade
Case Is >= 70: lblLetter.Text = "Pasing"
Case Else: lblLetter.Text = "Failing"
End Select
```

Bài thực hành 4

Dữ liệu cơ sở

Tóm tắt

Bài này chỉ bạn cách Visual Basic lưu giá trị dữ liệu ở nhiều dạng khác nhau. Các kiểu dữ liệu khác nhau cho phép bạn phân loại dữ liệu. Chẳng hạn, khi bạn làm việc với tiền bạc, sử dụng kiểu dữ liệu tiền tệ để Visual Basic đảm bảo chính xác tới 2 vị trí thập phân.

Khóa để tổ hợp dữ liệu trong chương trình Visual Basic là biến. Biến là tên vùng nhớ lưu trữ trong bộ nhớ mà bạn đã định nghĩa với lệnh Dim. Lệnh Dim yêu cầu Visual Basic dành riêng vùng nhớ, gán một kiểu dữ liệu cho biến, và đặt tên biến. Sau khi bạn định nghĩa biến, lệnh gán sẽ lưu giá trị vào biến. Trước khi bạn lưu giá trị vào biến, biến số lưu giá trị 0 và biến chuỗi lưu chuỗi rỗng.

Sau khi biến nhận giá trị, lệnh If và Select Case sẽ xác định đường dẫn chương trình thích hợp để lấy. Lệnh If và Select Case sẽ phân tích dữ liệu trong biến và thực hiện các quyết định trong thời gian thi hành dựa trên giá trị đó.

Trong chương này, bạn đã tìm hiểu:

- Cách phân biệt các kiểu dữ liệu
- Lý do phải định nghĩa và đặt tên biến
- Cách sử dụng các toán tử toán học của Visual Basic
- Khi độ ưu tiên toán tử trở nên quyết định
- Những gì phải làm khi bạn muốn vượt qua độ ưu tiên toán tử
- Cách mã lệnh If–Else và Select Case đưa ra quyết định
- Lý do tồn tại các tùy chọn Case khác nhau

Mô tả chương trình

Inventory Calculation

Inventory Calculation

Units Sold:

Price per Unit:

Discount Code:
1: 5%, 2: 10%,
3: 15%, 4: 20%

Calculate Inventory

Extended Amount:

Exit

Hình P4.1. *Màn hình mở project.*

Hình P4.1 minh họa ứng dụng PROJECT4.VBP ngay khi nạp và chạy chương trình. Mẫu biểu Project có nhiều điều khiển. Hãy lưu ý về các điều khiển sau trên mẫu biểu.

Người dùng phải nhập giá trị trong 3 hộp nhập phù hợp với số lượng hàng bán, đơn giá mặt hàng và mã khấu trừ đặc biệt. Người dùng phải nhập 1, 2, 3, hoặc 4 để cho biết phần trăm khấu trừ trên tổng số trị giá. Lệnh Calculate Inventory tính toán giá trị tiền dựa trên các giá trị người dùng nhập vào những hộp nhập và hiển thị kết quả trong nhãn bên dưới tên Entended Amount. Nút lệnh Exit sẽ dừng chương trình.

Chương trình hoạt động giống máy tính tiền đơn giản tính tổng số tiền hàng kiểm kê được có trừ đi số tiền chiết khấu dựa trên mã chiết khấu. Có nhiều cách cung cấp danh sách chiết khấu như thế trong Visual Basic, và chương trình này chỉ minh họa một phương pháp. Trong các bài sau, bạn sẽ học thêm điều khiển nút tùy chọn và cuốn hộp danh sách để hỗ trợ các mã chiết khấu phần trăm.

Ghi chú

Giả sử công ty chỉ cung cấp 4 mã chiết khấu được mô tả. Nếu người dùng để trống mã chiết khấu, chương trình sẽ không tính chiết khấu. Nếu người dùng nhập một mã chiết khấu, thật quá dễ để người dùng quên nhập dấu chấm thập phân khi muốn nhập giá trị chiết khấu thập phân như .15. Người dùng cũng có thể nhập vào dấu %, điều đó sẽ làm cho chương trình bối rối. Vì vậy, bảng mã chiết khấu là phương pháp rất tốt cho quá trình nhập giá trị phần trăm và loại trừ nhiều lỗi nhập liệu của người dùng.

Hoạt động chương trình

Nhập vào các giá trị sau trong điều khiển hộp nhập:

Units Sold: **20**

Price Per Unit: **2.75**

Discount Code: **3**

Nhấn nút lệnh Calculate Inventory (Alt+C). Visual Basic sẽ tính tổng trị giá hàng kiểm kê và hiển thị kết quả 46.75 ở bên phải nhãn Extended Amount.

Mặc dù project này làm việc với kiểu dữ liệu tiền tệ, song kết quả hiển thị không phải luôn có 2 số lẻ. Chẳng hạn, nhập vào đơn giá là 2.00, chúng ta có tổng giá trị tiền là 34 mà không có số lẻ nào. Visual Basic không bao giờ gánh vác cách hiển thị dữ liệu số theo ý bạn. Ngay cả với giá trị tiền tệ, Visual Basic cũng không hiển thị 2 số lẻ trừ khi bạn yêu cầu chính xác 2 số lẻ. Chương 7 mô tả cho bạn cách định dạng kết quả số xuất ra chính xác như bạn muốn. Còn bây giờ đừng quan tâm đến điều đó trong chương trình này. Điều quan trọng là hãy tập trung vào cách xuất kết quả, chứ không phải là cách định dạng kết quả.

Kiểm tra mã chiết khấu

Một trong những điểm nổi bật của ứng dụng PROJECT4.VBP là công dụng lệnh If trong thủ tục biến cố LostFocus của hộp nhập mã chiết khấu. Biến cố LostFocus xảy ra khi người dùng di chuyển tiêu điểm (focus) từ hộp soạn mã chiết khấu sang điều khiển khác. Vì vậy, thủ tục biến cố txtDisc_LostFocus() được minh họa trong Ví dụ P4.1, sẽ thi hành ngay lập tức sau khi người dùng nhập giá trị mã chiết khấu.

Ghi chú

Dòng 5 sử dụng thành phần Visual Basic được gọi là phương thức, mà bạn chưa gặp. Phương thức làm việc giống như hàm có sẵn, như Val() chẳng hạn. Tuy nhiên, thay vì chuyển dữ liệu, phương thức sẽ thực hiện hành động cho một điều khiển riêng. Bạn phải yêu cầu phương thức bằng cách xác định điều khiển, tiếp sau là dấu chấm và tên của phương thức.

Ví dụ P4.1. *Đảm bảo người dùng nhập vào một mã chiết khấu đúng*

```
1: Sub txtDisc_LostFocus ()
2: If (Val(txtDisc.Text) < 0) Or (Val(txtDisc.Text) > 4) Then
3: Beep
4: txtDisc.Text = ""
5: txtDisc.SetFocus
6: End If
7: End Sub
```

1. Thuộc tính Name của hộp nhập chiết khấu có nội dung là txtDisc, nên tên thủ tục biến cố LostFocus là txtDisc_LostFocus(). Dùng hàm Val() để chuyển giá trị hộp nhập thành một số trong khi nó kiểm tra để chắc chắn người dùng đã nhập vào một số từ 0 tới 4.
2. Thân lệnh If chỉ thi hành nếu người dùng nhập vào mã chiết khấu sai. Lệnh Beep sẽ phát tiếng bíp ra loa máy tính giúp người dùng chú ý.
3. Dòng này xóa bất kỳ giá trị nào người dùng nhập vào hộp nhập.
4. Dòng này trả tiêu điểm lại hộp nhập để người dùng phải tiếp tục nhập một giá trị đúng trước khi thực hiện bất kỳ điều gì khác.

1: Dòng này kết thúc lệnh If.

2: Dòng này dừng thủ tục biến cố.

3: Người dùng có thể nghe rõ cảnh báo về lỗi.

Tính tổng giá trị kiểm kê

Khi người dùng nhấp nút lệnh Calculate Inventory, thủ tục biến cố Click, được minh họa trong Ví dụ P4.2, sẽ thi hành. Thủ tục biến cố này sử dụng tổ hợp biến và lệnh Select Case để tính tổng số tiền đúng dựa trên các giá trị người dùng nhập vào hộp nhập với mức chiết khấu xác định.

Ví dụ P4.2. *Tính tổng giá trị kiểm kê.*

```
1: Sub cmdInven_Click ()
2: Dim Discount As Single
3: Dim ExtAmount As Currency
4: ExtAmount = Val(txtUnits.Text) * Val(txtPrice.Text)
5: Select Case Val(txtDisc.Text)
6: Case 0: Discount = 0
7: Case 1: Discount = .05
8: Case 2: Discount = .1
9: Case 3: Discount = .15
10: Case 4: Discount = .2
11: End Select
12: lblExt.Caption = ExtAmount – (Discount * ExtAmount)
13: End Sub
```

1. Thuộc tính Name của nút lệnh có giá trị là cmdlnven, cho nên tên của thủ tục biến cố Click là cmdlnven_Click().
2. Biến chính xác đơn Discount được định nghĩa để lưu kết quả tính trung gian.
3. Biến tiền tệ ExtAmount được định nghĩa để lưu kết quả tính trung gian.
4. Phần đầu tiên tổng số tiền kiểm kê được tính bằng cách nhân đơn giá bán với tổng số lượng mặt hàng bán.
5. Select Case đưa ra quyết định dựa trên một trong bốn giá trị được lưu trong hộp nhập txtDisc.
6. Nếu mã chiết khấu là 0, phần trăm chiết khấu bằng 0.
7. Nếu mã chiết khấu là 1, phần trăm chiết khấu bằng 5%

8. Nếu mã chiết khấu là 2, phần trăm chiết khấu bằng 10%
9. Nếu mã chiết khấu là 3, phần trăm chiết khấu bằng 15%.
10. Nếu mã chiết khấu là 4, phần trăm chiết khấu bằng 20%.
11. Kết thúc lệnh Select Case.
12. Dòng này hoàn thành quá trình tính tổng số chiết khấu bằng cách áp dụng chiết khấu vào tổng trị giá tính được trong biến ExtAmount, và nó hiển thị tổng giá trị hàng cuối cùng trên mẫu biểu.

4: Luôn đổi các giá trị điều khiển thành số bằng cách dùng hàm Val() trước khi bạn tính các giá trị.

11: Không có Case Else *được* yêu cầu bởi vì thủ tục txtDisc LostFocus() đảm bảo rằng chỉ những giá trị hợp lệ mới được xuất hiện trong txtDisc.

Đóng trình ứng dụng

Bài thực hành này giúp cô đọng kiến thức của bạn về cách sử dụng, cũng như thời điểm sử dụng biến dữ liệu. Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Bài kế tiếp bổ sung các kỹ năng lập trình cho bạn và tăng thêm khả năng cho chương trình bạn thiết kế.

Chương V

<u>*Bài 9*</u>

Các ghi chú và hộp thông báo

- **Công dụng của ghi chú**
- **Mã lệnh chú thích**
- **Giới thiệu về hộp thông báo và hộp nhập dữ liệu**
- **Lệnh MsgBox**
- **Hàm MsgBox()**
- **Các hàm InputBox()**

Visual Basic có nhiều chương trình, không ít điều khiển Visual, và mã lệnh. Thông thường, bạn sẽ đưa các thông báo được gọi là *ghi chú* (remark) vào những chương trình mà Visual Basic, Windows, và máy tính của bạn hoàn toàn bỏ qua. Các ghi chú không dành cho máy tính, chúng dành cho lập trình viên.

Vào một thời điểm nào đó, bạn cần hiển thị thông báo đến người dùng và nhận câu trả lời từ người dùng. Không có loại điều khiển nào hoặc đối tượng Visual Basic nào làm việc tốt hơn là *hộp thông báo* (message box) và *hộp nhận thông tin dữ liệu từ người dùng* (input box). Bài này sẽ chỉ bạn cách hiển thị và quản lý các hộp này.

CÔNG DỤNG CỦA GHI CHÚ

Khái niệm

Các ghi chú (remark) sẽ giúp bạn lẫn lập trình viên khác có thể sửa đổi và cập nhật các ứng dụng Visual Basic trong tương lai. Ghi chú cung cấp thông báo mô tả được giải thích bằng tiếng Anh (hay bất kỳ ngôn ngữ nào bạn thích) những gì sẽ tiếp tục trong các thủ tục biến cố của chương trình.

Điều đó nói lên rằng một chương trình được viết một lần và đọc nhiều lần. Câu nói đó không sai nếu xét về bản chất của những ứng dụng. Lấy ví dụ, bạn sẽ viết một chương trình giúp đỡ bạn hay nhà kinh doanh tính toán các phép tính được yêu cầu và duy trì việc tìm kiếm các giao dịch hàng ngày. Theo thời gian, các yêu cầu sẽ thay đổi. Những nhà kinh doanh mua và bán với các nhà kinh doanh khác, chính phủ sẽ thay đổi yêu cầu về thuế và nhu cầu của con người sẽ thay đổi. Bạn nên hiểu rằng, sau khi viết và thực hiện một chương trình, bạn sẽ sửa đổi chương trình đó về sau. Nếu sử dụng chương trình trong kinh doanh, chắc chắn bạn phải thực hiện nhiều sửa đổi ở chương trình để làm mới các điều kiện.

Khái niệm mới

Bảo trì chương trình là thay đổi những chương trình đã sử dụng quá hạn.

Ghi chú

Nếu bạn lập trình cho một ai khác, khả năng người đó sẽ sửa đổi chương trình bạn viết rất cao cũng như bạn sẽ sửa đổi các chương trình do những lập trình viên khác viết. Vì vậy, khi bạn viết chương trình, hãy nghĩ đến việc bảo trì mà sau này bạn và người khác sẽ thực hiện. Hãy viết chương trình rõ ràng, bằng cách sử dụng các khoảng trống rộng và định lề, thêm chú thích giải thích các phần khó của mã lệnh.

Một ghi chú là một thông báo mà bạn sẽ đặt vào bên trong mã lệnh chương trình. Các lập trình viên quan tâm đến việc bảo trì biết rằng một ghi chú dài làm mã lệnh sáng sủa hơn và giúp đỡ công việc bảo trì sau này. Visual Basic bỏ qua hoàn toàn tất cả chú thích bởi vì các chú thích này cho người ta xem trong mã lệnh chương trình. Người dùng không xem ghi chú bởi vì họ không thấy mã lệnh của chương trình, hiếm khi người dùng thấy dữ liệu xuất ra của một chương trình.

Các lập trình viên thường thêm ghi chú vào chương trình của họ vì những mục đích sau:

- Nói rõ tên của lập trình viên và ngày mà chương trình được viết
- Mô tả mục đích chung của chương trình
- Mô tả ở đầu các thủ tục mục đích chung của thủ tục đó

- Giải thích những câu lệnh khó hoặc phức tạp để lập trình viên khác sửa đổi chương trình sau này có thể hiểu các dòng mã lệnh mà không phải đoán mã lệnh khó hiểu

Ghi chú

Ngay cả khi tự bạn viết chương trình, và nếu bạn là người duy nhất sẽ sửa đổi chương trình của bạn, bạn vẫn nên thêm các ghi chú vào chương trình! Hàng tuần hoặc hàng tháng sau khi viết một chương trình, bạn sẽ quên các chi tiết chính xác của chương trình và ghi chú mà bạn đặt rải rác quanh mã lệnh sẽ đơn giản hóa việc bảo trì của bạn và sẽ giúp bạn tìm thấy mã lệnh cần thay đổi.

Lời nhắc

Hãy thêm các ghi chú khi bạn viết chương trình. Thông thường, các lập trình viên tự nói với họ "Tôi sẽ hoàn thành chương trình và thêm các chú thích sau". Sự thật các ghi chú không được thêm. Chỉ sau này khi lập trình viên cần sửa đổi chương trình, lúc đó họ mới chú ý đến việc thiếu các chú thích và hối tiếc về điều đó.

Ôn lại

Hãy thêm chú thích vào chương trình để bạn và người khác có thể hiểu nhanh chóng bản chất của chương trình và dễ dàng sửa đổi khi cần thiết.

MÃ LỆNH CHÚ THÍCH

Khái niệm

Visual Basic hỗ trợ nhiều dạng chú thích. Không giống một số ngôn ngữ lập trình khác, rất dễ thêm chú thích Visual Basic vào mã lệnh của bạn, và bản chất dạng tự do của nó cho phép bạn thêm chú thích vào bất cứ khi nào và tới bất cứ nơi đâu cần chúng.

Visual Basic hỗ trợ hai loại chú thích sau:

- Chú thích bắt đầu với lệnh Rem
- Chú thích bắt đầu với dấu nháy đơn (')

Lệnh Rem tỏ ra giới hạn hơn dấu nháy đơn và không dễ sử dụng như dấu nháy đơn. Tuy nhiên, bạn sẽ đọc qua các chương trình sử dụng lệnh Rem, vì thế bạn nên học cách làm việc của lệnh Rem. Sau đây là dạng thức lệnh Rem:

```
Rem The remark's text
```

Bạn có thể đặt bất cứ điều gì bạn muốn vào vị trí văn bản của chú thích. Vì vậy, tất cả dòng sau đều là các chú thích:

```
Rem Programmer: Bob Enyart, Date: Mar-27-1996
Rem This program supports the check-in and check-out
Rem process for the dry-cleaning business.
Rem This event procedure executes when the user
Rem clicks on the Exit command button. When pressed,
Rem this event procedure closes the program's data
Rem files, prints an exception report, and terminates
Rem the application
```

Dòng chú thích đầu tiên chứa tên của lập trình viên và ngày chương trình được cập nhật lần cuối. Nếu một ai khác phải sửa đổi chương trình sau đó, họ có thể tìm đến lập trình viên ban đầu nếu cần để hỏi về mã lệnh chương trình. Chú thích thứ hai mô tả mục đích chung của chương trình bằng việc nêu rõ mục đích chương trình ở mức cao nhất. Chú thích thứ ba có thể xuất hiện ở đầu một thủ tục biến cố Click của nút lệnh. Bạn có thể thêm một hay nhiều dòng chú thích phụ thuộc vào tổng số mô tả cần vào lúc đó trong chương trình. Visual Basic bỏ qua tất cả các dòng bắt đầu với Rem. Khi một ai đó đọc mã lệnh, người này sẽ biết ai là lập trình viên, ngày chương trình được viết, mục đích chung của chương trình, và mô tả chung của từng thủ tục kể cả phần chú thích.

Hãy nhớ rằng bạn đã sử dụng dấu nháy đơn thay cho vị trí của các lệnh Rem trong những chú thích trước. Các chú thích được viết lại sau đây minh họa rằng các chú thích ' thậm chí hiệu quả hơn bởi vì từ Rem có khi là văn bản của chú thích:

```
' Programmer: Bob Enyart, Date: Mar-27-1996
' This program supports the check-in and check-out
' process for the dry-cleaning business.
' This event procedure executes when the user
' clicks on the Exit command button. When pressed,
' this event procedure closes the program's data
' files, prints an exception report, and terminates
' the application
```

Chú thích không phải bắt đầu ở đầu các thủ tục biến cố. Bạn có thể đặt chú thích ở giữa dòng mã lệnh, như được thực hiện ở đây:

```
Dim Rec As Integer
Rem Step through each customer record
For Rec = 1 To NumCusts
 ' Test for a high balance
 If custBal(Rec) > 5000 Then
 Call PayReq
 End If
Next Rec
```

Chú thích

Đừng cố gắng hiểu chi tiết mã lệnh này. Bây giờ hãy tập trung vào các chú thích. Mã lệnh chứa một số đặc trưng nâng cao (mảng và thủ tục con Visual Basic) mà bạn sẽ làm quen với nó ở phần sau cuốn sách này.

Chú thích dấu nháy đơn thuận tiện hơn việc sử dụng Rem bởi vì bạn có thể đặt các chú thích ' ở cuối lệnh Visual Basic. Khi đặt một chú thích ở bên phải các dòng mã lệnh xác định, bạn có thể phân loại mục đích của mã lệnh. Hãy xem xét cách thức phần mã lệnh sau sử dụng chú thích để giải thích những dòng mã lệnh xác định:

```
a = 3.14159 * r ^ r ' Calculate a circle's area
```

Có lẽ chỉ nhà toán học mới có thể dịch công thức mà không cần chú thích. Chú thích giúp mọi người không phải là nhà toán học hiểu mục đích của lệnh. Không có lý do gì bạn phải thử nghiệm lại mã lệnh mỗi lần đọc mã lệnh. Bằng cách đọc chú thích, bạn có thể phát hiện ra mục đích của mã lệnh mà không mất thời gian dịch mã lệnh của Visual Basic.

Tuy nhiên, chú thích sai chẳng giúp gì cho việc làm rõ ràng mã lệnh, cho nên, đừng lạm dụng quá mức các chú thích. Vấn đề thực tế, nhiều dòng lệnh không cần chú thích để giải thích mục đích của chúng. Chú thích sau không cần thiết và vô giá trị đối với việc lập trình của bạn lẫn bất kỳ ai sau này bảo trì chương trình:

```
Dim Sales As Single ' Define a variable named Sales
```

Tóm tắt

Ví dụ 9.1 có chứa một thủ tục Select Case mà bạn đã thấy trong Ví dụ 8.5 của bài trước. Mã lệnh này bao gồm những chú thích giúp phân loại mục đích của mã lệnh.

Ôn lại

Hãy thêm chú thích vào một chương trình Visual Basic, bằng cách sử dụng hoặc lệnh Rem hoặc dấu ', để báo cho các lập trình viên khác biết rõ những gì chương trình đang thực hiện. Bổ sung thêm các chú thích khi bạn viết chương trình để chú thích cập nhật đúng ngày và luôn luôn làm mới mục đích của bạn.

Ví dụ 9.1 *Một số chú thích phân loại mục đích mã lệnh.*

```
Rem The following Select Case to End Select code
Rem assigns a student's grade and school name
Rem to the label on the form. The code checks
Rem to make sure that the student is not too
Rem young to be going to school.
Select Case Age
' Check for too young...
Case Is <5: lblTitle.Text = "Too young"
' Five-year olds are next assigned
Case 5: lblTitle.Text = "Kindergarten"
' Six to eleven...
Case 6 To 11: lblTitle.Text = "Elementary"
lblSchool.Text = "Lincoln"
' Twelve to fifteen...
Case 12 To 15: lblTitle.Text = "Intermediate"
lblSchool.Text = "Washington"
' Sixteen to eighteen
Case 16 To 18: lblTitle.Text = "High School"
lblSchool.Text = "Betsy Ross"
' Everyone else must go to college
Case Else: lblTitle.Text = "College"
lblSchool.Text = "University"
End Select
```

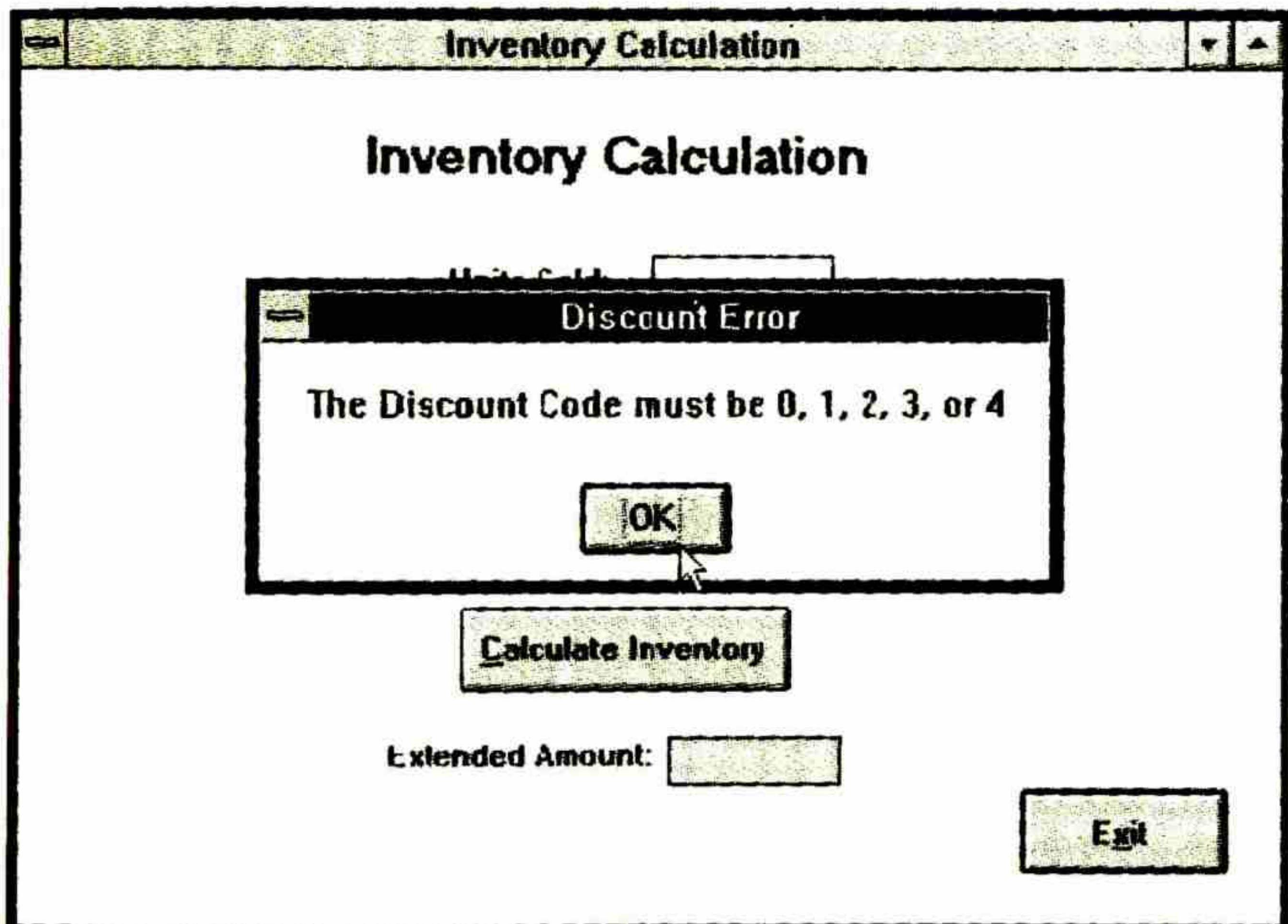
Phân tích

Chỗ thụt vào được sử dụng trong các chú thích thuộc Ví dụ 9.1 để đặt chú thích ở nơi hiệu quả nhất. Chú thích trên các dòng 7, 9, 11, 14, 17, và 20 không xuất hiện ở cuối dòng mã lệnh tương ứng của chúng bởi vì chú thích nằm quá xa bên phải và sẽ không phù hợp trong cửa sổ Code. Tuy nhiên, chú thích làm rõ ràng từng lựa chọn Case. Hãy chú ý rằng một vài dòng lệnh đầu tiên, từ 1 đến 5, mô tả chung về việc tính toán mã lệnh sau đó.

GIỚI THIỆU VỀ HỘP THÔNG BÁO VÀ HỘP NHẬP DỮ LIỆU

Khái niệm mới

Message box – hộp thông báo ***thực hiện công việc xuất dữ liệu chương trình đặc biệt.***



Hình 9.1. *Một hộp thông báo có thể cung cấp cho người dùng thông tin có ích.*

Trong các chương trình sẽ không ít lần bạn cần hỏi người dùng hay muốn hiển thị thông báo lỗi và lời khuyên tới người dùng. Thông thường, các điều khiển trên mẫu biểu làm việc không tốt với những hộp thoại như thế. Ví dụ, trong Bài Thực Hành ở Chương 4, người dùng phải nhập vào số 0 đến 4 khi được hỏi về cách tính phần tràm khấu hao. Nếu người dùng đã nhập vào một mã lệnh sai, chương trình sẽ phát tiếng bíp và xóa nội dung nhập sai ấy, nhưng chương trình không báo cho người dùng biết lý do tại sao nội dung nhập vào bị lỗi. Người dùng không phải lúc nào cũng biết những gì được chờ đợi ở họ. Người dùng sẽ được giúp đỡ nhiều bằng cách xem thông báo lỗi, như thông báo trong Hình 9.1, trước khi chương trình xóa nội dung sai của người dùng.

Ghi chú

Hộp thông báo (message box) không phải là điều khiển. Khác với điều khiển ở trên mẫu biểu trong suốt chu kỳ thi hành toàn bộ chương trình, một hộp thông báo vận hành ở trước mẫu biểu và biến mất khi người dùng trả lời hộp thông báo, thường bằng việc nhấp nút lệnh OK của hộp thông báo.

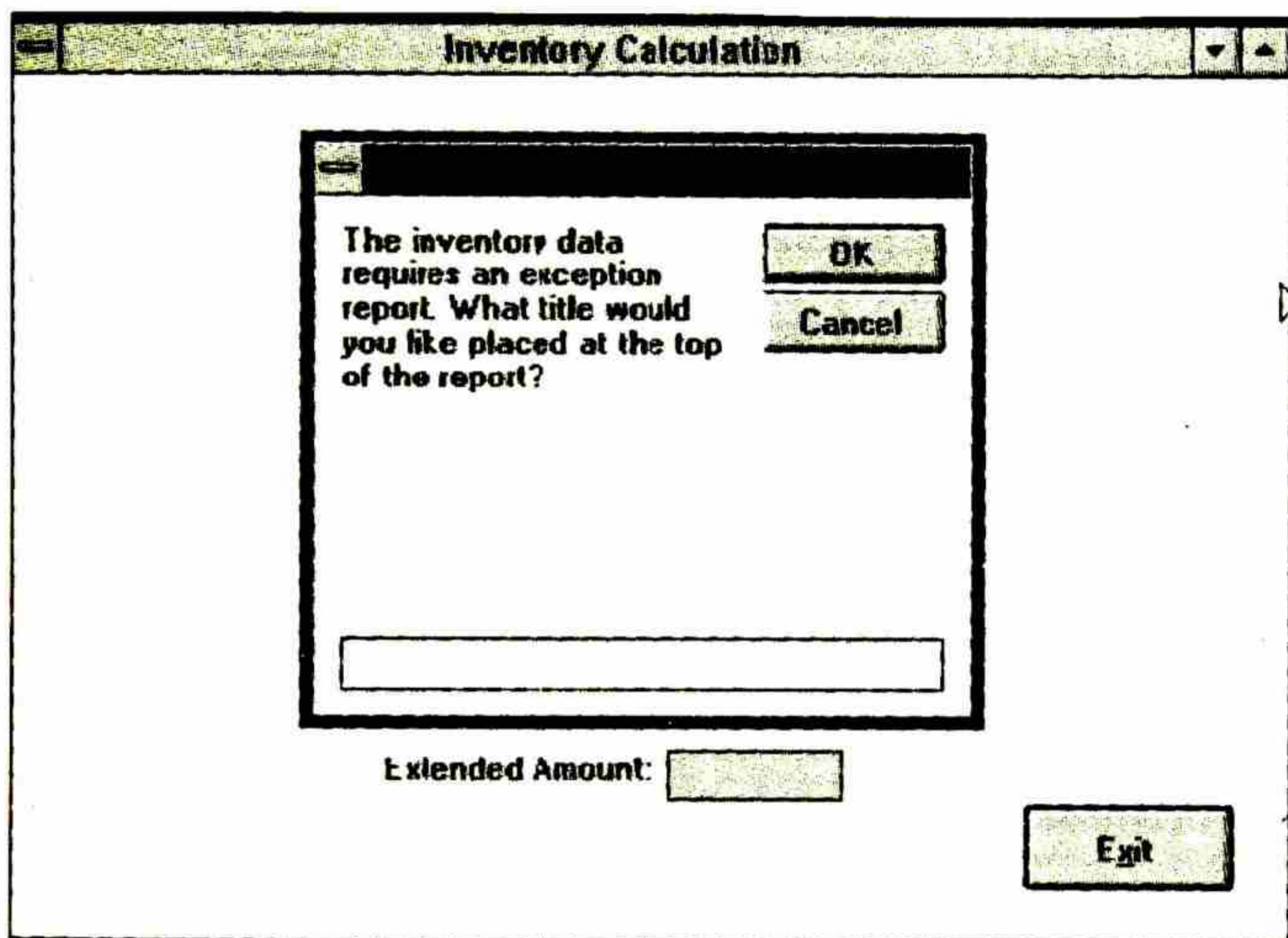
Có hai cách đưa ra *hộp thông báo* (message box). Bạn có thể sử dụng lệnh MsgBox hay hàm MsgBox(). Bạn đã thấy một hàm xây dựng sẵn trước đây, đó là hàm Val(), dùng để đổi một chuỗi số thành một số. Visual Basic bao gồm nhiều hàm xây dựng sẵn; bạn sẽ làm quen trong Chương 7. Tuy nhiên, trước mắt bạn cần học một vài hàm như là Val() và MsgBox(), bởi vì chúng sẽ giúp bạn làm việc với lệnh. Việc chọn sử dụng lệnh MsgBox hay hàm MsgBox() phụ thuộc vào sự trả lời mà bạn cần người dùng thực hiện để hiển thị hộp thông báo.

Khái niệm mới

Input box - hộp nhận thông tin từ người dùng ***thực hiện công việc nhận dữ liệu chương trình đặc biệt.***

Các điều khiển hộp nhập, bạn đã thấy, nhận phần lớn giá trị từ người dùng. Những điều khiển khác, bạn sẽ học khi đọc cuốn sách này, cũng nhận dữ liệu nhập vào của người dùng từ bàn phím hay mouse. Tuy nhiên, các điều khiển Visual Basic lại không đủ để xử lý tất cả dữ liệu vào mà chương trình của bạn sẽ cần. Khi người dùng phải trả lời các loại câu hỏi xác định, tốt hơn cả nên sử dụng hộp nhận thông tin

dữ liệu từ người dùng (input box). Hộp nhập và những điều khiển khác chỉ tốt cho việc nhận dữ liệu cố định từ người dùng, chẳng hạn như giá trị dữ liệu mà chương trình sẽ tính toán. Hộp nhận dữ liệu (input box) thật tốt trong việc hỏi người dùng những câu hỏi chỉ phát triển trong điều kiện xác định. Các hộp nhận dữ liệu (input box) luôn cho người dùng một nơi để trả lời. Trong Hình 9.2, hộp nhận dữ liệu (input box) đang hỏi người dùng về một tiêu đề sẽ hiện ra ở đầu danh sách báo cáo được in trên giấy.



Hình 9.2. *Sử dụng hộp nhận dữ liệu để hỏi thông tin.*

Lưu ý rằng có nhiều cách cho người dùng trả lời hộp nhận dữ liệu trong Hình 9.2. Người dùng có thể trả lời câu hỏi bằng cách nhập tiêu đề ở cuối hộp nhận dữ liệu và nhấn phím Enter hay nhấp nút lệnh OK. Người dùng cũng có thể nhấp nút lệnh Cancel cho biết họ đã không nhập tiêu đề. Vì vậy, chương trình phải có khả năng đọc câu trả lời được nhập vào của người dùng. Việc trả lời các nút lệnh trong hộp thông báo và hộp nhận dữ liệu bạn sẽ học trong những phần còn lại của chương này.

Ôn lại

Message box hiển thị dữ liệu xuất ra và Input box nhận giá trị nhập vào. Hộp thông báo và hộp nhận dữ liệu cung cấp nhiều phương cách cho chương trình hỏi thông tin mà các điều khiển thông thường không thể xử lý. Bên cạnh đó việc sử dụng các điều khiển để hiển thị và lấy giá trị dữ liệu luôn luôn cần đến. Sử dụng message box và input box để hiển thị thông báo và nhận câu trả lời trong trường hợp đặc biệt, như điều kiện lỗi và xử lý biệt lệ.

LỆNH MsgBox

Khái niệm

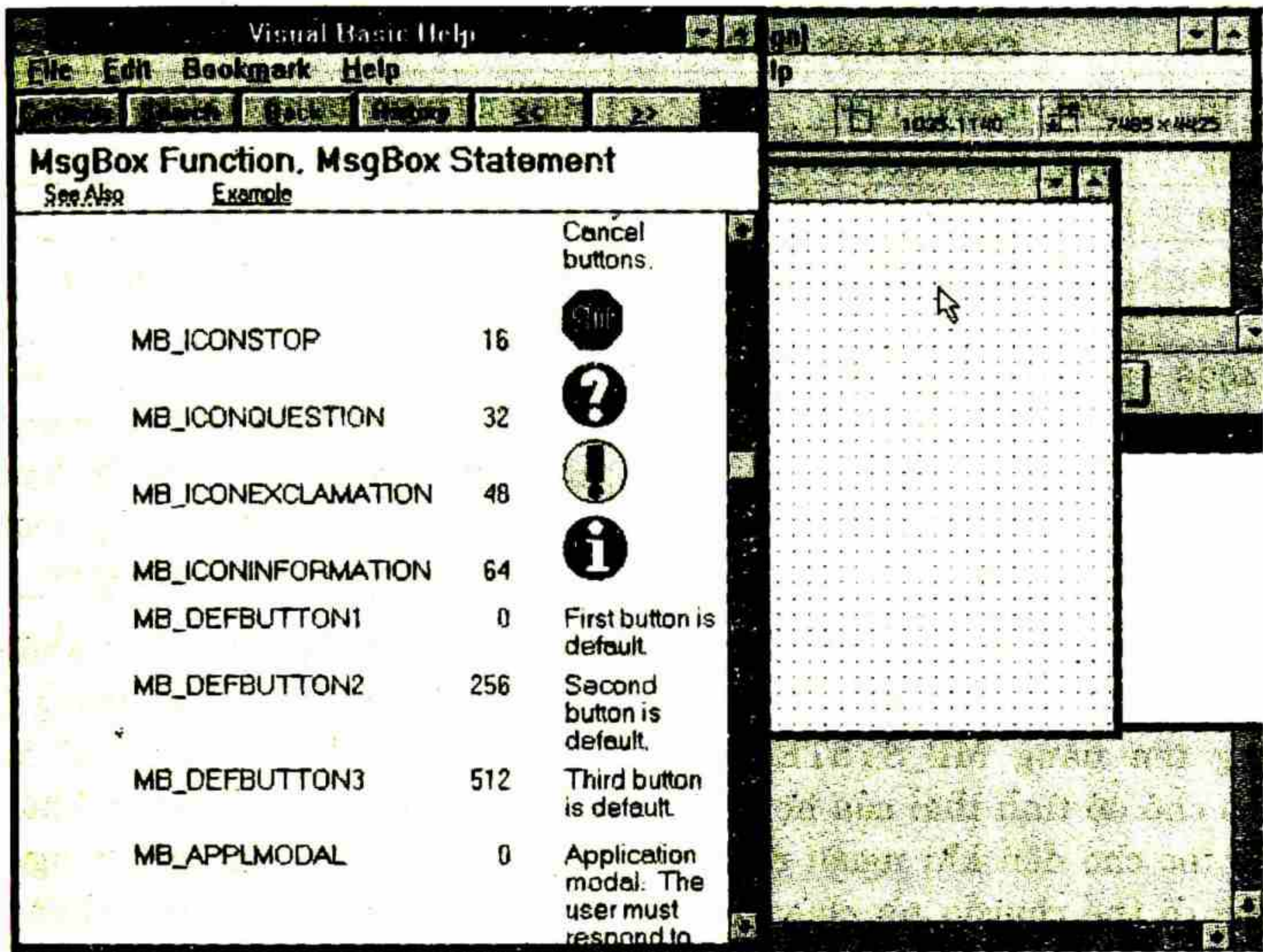
Lệnh MsgBox hiển thị thông báo cho người dùng. Thêm vào đó, hàm MsgBox() còn giúp chương trình hiển thị và kiểm tra nhiều thao tác nhấp nút lệnh trên cửa sổ hộp thông báo. Phần này giải thích cách sử dụng lệnh MsgBox.

Tất cả *hộp thông báo* (message box) hiện ra bằng lệnh MsgBox sẽ hiển thị một nút lệnh OK. Nút lệnh cho người dùng cơ hội đọc thông báo. Khi người dùng đọc xong thông báo, họ có thể nhấp OK để thoát ra khỏi hộp thông báo. Chương trình của bạn sẽ đình chỉ việc thi hành trong suốt quá trình đọc hộp thông báo. Vì vậy, lệnh theo sau lệnh MsgBox sẽ thi hành khi người dùng nhấp OK.

Sau đây là dạng của lệnh MsgBox:

MsgBox *msg* [, [*type*] [, *title*]]

msg là một chuỗi (hoặc là biến hoặc là hằng chuỗi bao trong cặp ngoặc kép) và các dạng văn bản của thông báo hiển thị trong hộp thông báo. *type* là giá trị số hay biểu thức tùy chọn mô tả các tùy chọn bạn muốn trong hộp thông báo. Hình 9.3 cho thấy những biểu tượng nào bạn có thể đặt trong một hộp thông báo (Visual Basic hiển thị không có biểu tượng nếu bạn không xác định một giá trị *type*). *title* là một chuỗi tùy chọn chuyên trình bày văn bản trong thanh tiêu đề của hộp thông báo. Nếu bạn bỏ qua *title*, Visual Basic sẽ sử dụng tên của project cho nội dung thanh tiêu đề của hộp thông báo.



Hình 9.3. *Các biểu tượng bạn có thể hiển thị trong một hộp thông báo.*

Khái niệm mới

Modality – chế độ tình thái *xác định người dùng hay hệ thống sẽ trả lời một hộp thông báo.*

Các tùy chọn bạn chọn, bằng việc sử dụng giá trị *type* trong lệnh MsgBox, xác định hộp thông báo sẽ hiển thị một biểu tượng như các điều khiển nhưng được *chế độ tình thái* (modality) của hộp thông báo quyết định. Bảng 9.1 chứa các giá trị mã lệnh mà bạn sẽ sử dụng để định giá trị kiểu MsgBox.

Bảng 9.1. *Các giá trị type của lệnh MsgBox*

Giá trị	Giá trị hằng CONSTANT.BAS	Diễn giải
16	MB_ICONSTOP	Hiển thị biểu tượng STOP
32	MB_ICONQUESTION	Hiển thị biểu tượng dấu ?
48	MB_ICONEXCLAMATION	Hiển thị dấu cảm thán
64	MB_ICONINFORMATION	Hiển thị biểu tượng chữ thường (nghĩa là information)
4096	MB_SYSTEMMODAL	Ứng dụng của người dùng là hệ thống modal, nghĩa là hộp thông báo phải được xử lý trước khi bạn có thể chuyển tới bất kỳ một chương trình Windows nào khác

Modality (chế độ tình thái) thường gây nhầm lẫn. Nếu bạn không xác định giá trị hệ thống kiểu model là 4096 (hay nếu bạn không sử dụng tên hằng MB_SYSTEMMODAL trong CONSTANT.BAS để xác định chế độ tình thái của hệ thống), ứng dụng của người dùng sẽ không tiếp tục cho đến khi người dùng đóng hộp thông báo. Tuy nhiên người dùng có thể chuyển tới chương trình Windows khác bằng cách nhấn tổ hợp phím Alt+Tab hay chuyển tới một chương trình khác qua việc sử dụng menu điều khiển của ứng dụng. Tuy nhiên, nếu bạn xác định rằng hộp thông báo là hệ thống modal, người dùng sẽ không thể chuyển tới chương trình Windows khác cho đến khi người dùng trả lời hộp thông báo, bởi vì hộp thông báo sẽ có điều khiển đầy đủ của hệ thống. Trái lại các hộp thông báo hệ thống modal cho những báo lỗi nghiêm trọng thì người dùng phải đọc và trả lời trước khi tiếp tục chương trình.

Ghi chú

Nếu bạn không xác định một biểu tượng, Visual Basic sẽ không hiển thị biểu tượng. Còn như bạn không xác định chế độ tình thái của hệ thống, Visual Basic cho rằng bạn muốn một ứng dụng modal trong hộp thông báo.

Tóm tắt

Ví dụ 9.2 giúp trả lời các câu hỏi mà bạn thắc mắc về những hộp thông báo. Ví dụ bao gồm một dãy lệnh message box. Mỗi lệnh sẽ hiển thị một hộp thông báo khác nhau.

Ôn lại

Sử dụng lệnh MsgBox để hiển thị thông báo, chẳng hạn như thông báo lỗi đến người dùng. Các hộp thông báo sẽ có một nút OK, người dùng nhấp để đóng hộp thông báo. Phụ thuộc vào việc bạn xác định giá trị *type*, hộp thông báo có lẽ chứa một biểu tượng như chế độ tình thái (modality) của hệ thống. Nếu bạn xác định giá trị tiêu đề thứ ba, thanh tiêu đề của hộp thông báo sẽ chứa văn bản của tiêu đề do bạn đề nghị.

Ví dụ 9.2. *Hiển thị các hộp thông báo khác nhau.*

```
' A simple message box with a message and no icon
MsgBox "Just a message"
' A message box with the stop sign icon
MsgBox "Stop in the name of love", MB_ICONSTOP
' A message box with the question mark icon
MsgBox "Did you turn on the printer?", MB_ICONQUESTION
' A message box with the exclamation icon
MsgBox "Don't forget to exit!", MB_ICONEXCLAMATION
' A message box with the "i" information icon as
' well as a title that's not the project name
MsgBox "A byte is 8 bits", MB_ICONINFORMATION, "A title"
' A message box that is system modal
MsgBox "You cannot switch programs", MB_SYSTEMMODAL
' A message box with a question mark icon, a system modal setting,
' and a title of Title
MsgBox "Info icon, system modal, and a title",
MB_ICONQUESTION + MB_SYSTEMMODAL, "Title"
```

Ví dụ 9.2 minh họa nhiều loại hộp thông báo khác nhau. Giả sử tên project là PROJECT1.VBP, Hình 9.4 cho thấy thông báo đơn giản sẽ xuất hiện từ lệnh MsgBox của dòng thứ 2.



Hình 9.4. *Một hộp thông báo đơn giản không có biểu tượng hay tiêu đề.*

Phân tích

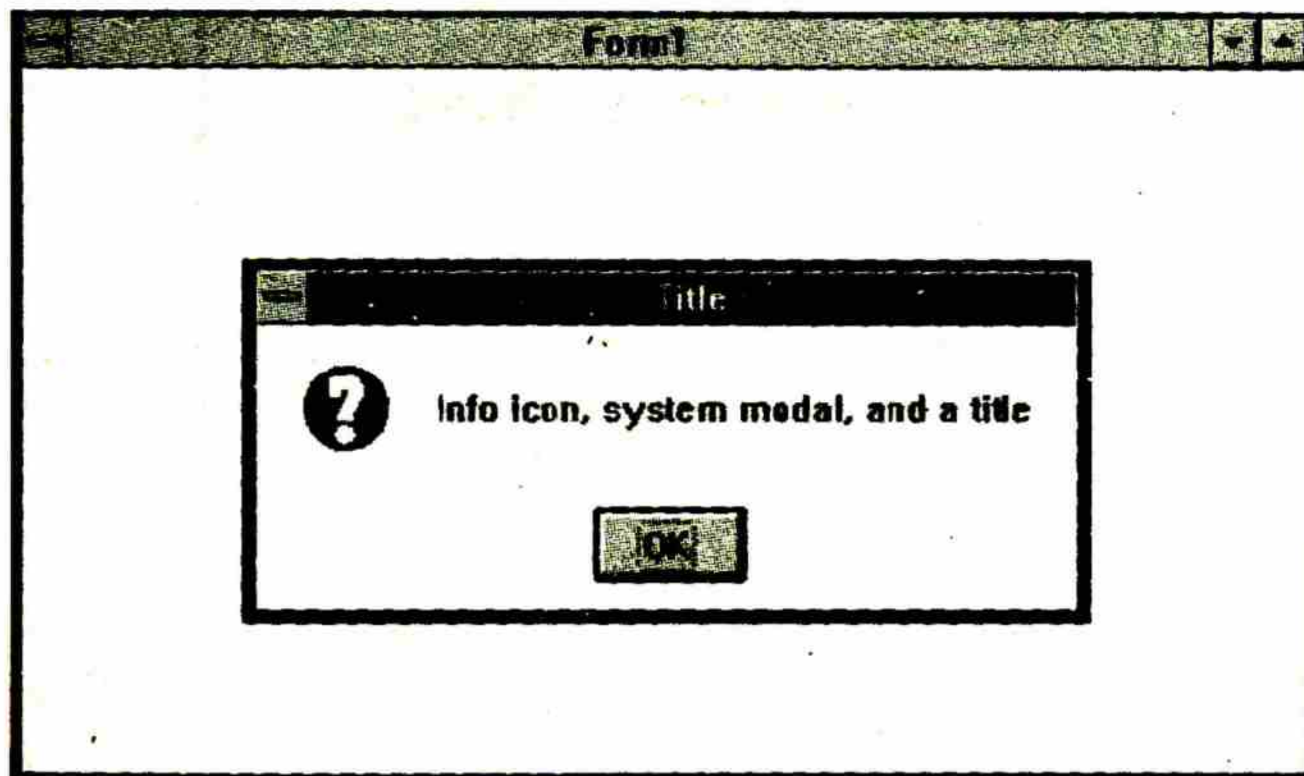
Để ý sẽ thấy không có biểu tượng bởi vì dòng 2 không chứa giá trị *type*. Cũng như vậy, tiêu đề của hộp thông báo là Project1 bởi vì tên của project là PROJECT1.VBP và lệnh MsgBox không xác định một tiêu đề khác. Hộp thông báo cũng là ứng dụng modal. Người dùng phải trả lời thông báo trước khi tiếp tục chương trình. Tuy nhiên, người dùng có thể chuyển tới chương trình Windows khác, bằng cách nhấn tổ hợp phím Alt+Tab, hay tổ hợp phím Alt+spacebar, hay chọn từ menu điều khiển.

Lệnh MsgBox trong dòng 4 hiển thị một biểu tượng STOP ở bên trái thông báo. Ít được sử dụng hơn so với việc sử dụng tên biến hằng trong CONSTANT.BAS, nhưng bạn vẫn có thể nhập 16 cho giá trị *type*.

Lệnh MsgBox của dòng 6 sẽ hiển thị biểu tượng dấu ? ở bên trái thông báo. Lệnh MsgBox của dòng 8 sẽ hiển thị biểu tượng dấu cảm thán ở bên trái thông báo.

Dòng 11 sẽ hiển thị một thông báo với biểu tượng information. Biểu tượng information là một biểu tượng dễ thương khi đưa ra lời khuyên cho người dùng. Dòng 11 cũng hiển thị một tiêu đề trong thanh tiêu đề của hộp thông báo. Một tiêu đề đơn giản: A title.

MsgBox của dòng 13 đưa ra một chế độ tình thái của hệ thống. Lệnh MsgBox của dòng 14 sẽ hiển thị một thông báo, một biểu tượng information, và một tiêu đề là a title. Hình 9.5 sẽ chỉ kết quả xuất của hộp thông báo dòng 13.



Hình 9.5. *Một hộp thông báo đầy đủ đưa ra thông báo, biểu tượng và tiêu đề.*

Ghi chú

Lưu ý rằng Visual Basic sẽ mở rộng hoặc thu hẹp hộp thông báo để chứa văn bản đầy đủ mà bạn muốn hiển thị.

Lời nhắc

Nếu bạn muốn xác định tiêu đề trong một hộp thông báo mà không hiển thị biểu tượng hay thay đổi chế độ tình thái của hệ thống, hãy chèn hai dấu phẩy trước chuỗi tiêu đề, như sau:

```
MsgBox "A byte is 8 bits", , "A title"
```

HÀM MsgBox()

Khái niệm

Sử dụng hàm MsgBox() khi bạn muốn người dùng chỉ định một trả lời tới thông báo trong hộp thông báo. Bằng cách sử dụng hàm MsgBox(), bạn có thể hiển thị nhiều nút lệnh khác nhau bên trong hộp thông báo và xác định nút lệnh nào người dùng đã nhấn để bạn biết cách người dùng trả lời thông báo.



Hình 9.6. *Các hộp thông báo có thể có nhiều nút lệnh.*

Hình 9.6 chỉ ra một hộp thông báo trông hơi khác so với các hộp thông báo khác mà bạn đã thấy trước đó. Vì một nút lệnh OK, hộp thông báo của Hình 9.6 có chứa các nút lệnh. Hàm MsgBox() cho phép bạn viết chương trình các hộp thông báo với nhiều nút lệnh và sau đó xác định nút lệnh nào người dùng đã nhấn để đóng hộp thông báo.

Dạng thức hàm MsgBox() hầu như đúng với dạng thức lệnh MsgBox. Tuy nhiên, bạn phải sử dụng hàm MsgBox() khác so với lệnh, luôn luôn gán một hàm MsgBox() tới một biến. Sau đây là dạng thức hàm MsgBox():

```
anIntVariable = MsgBox( msg [, [type] [, title]])
```

Như bạn thấy, dạng thức hàm MsgBox() khác so với lệnh MsgBox ở chỗ bạn sẽ gán hàm MsgBox() cho một biến nguyên mà bạn đã định nghĩa. Thêm vào đó, hàm MsgBox() hỗ trợ nhiều giá trị *type* hơn lệnh MsgBox. Bảng 9.2 sẽ liệt kê các giá trị type của hàm MsgBox() kèm theo tên hằng tương ứng của chúng trong CONSTANT.BAS.

Bảng 9.2. *Các giá trị type của hàm MsgBox()*

Giá trị	Giá trị CONSTANT.BAS	Diễn giải
0	MB_OK	Nút OK xuất hiện duy nhất trong hộp thông báo.
1	MB_OKCANCEL	Các nút OK và Cancel sẽ xuất hiện.
2	MB_ABORTRETRYIGNORE	Các nút Abort, Retry, và Cancel sẽ xuất hiện.
3	MB_YESNOCANCEL	Các nút Yes, No, và Cancel sẽ xuất hiện.
4	MB_YESNO	Các nút Yes và No sẽ xuất hiện.
5	MB_RETRYCANCEL	Các nút Retry và Cancel sẽ xuất hiện.
16	MB_ICONSTOP	Hiển thị các biểu tượng Stop.
32	MB_ICONQUESTION	Hiển thị dấu ?
48	MB_ICONEXCLAMATION	Hiển thị dấu cảm thán.
64	MB_ICONINFORMATION	Hiển thị chữ thường i (nghĩa là thông tin).
0	MB_DEFBUTTON1	Nút đầu tiên sẽ nhận tiêu điểm (focus) ban đầu.
256	MB_DEFBUTTON2	Nút thứ hai sẽ nhận tiêu điểm ban đầu.
512	MB_DEFBUTTON3	Nút thứ ba sẽ nhận tiêu điểm ban đầu.
4096	MB_SYSTEMMODAL system message	Ứng dụng của người dùng là modal, nghĩa là hộp phải được xử lý trước khi bạn có thể chuyển tới một chương trình Windows bất kỳ nào khác.

Sau đây là hàm MsgBox() đã hiển thị hộp thông báo trong Hình 9.6:

```
BPress = MsgBox("Are you ready for the report?",
MB_ICONQUESTION + MB_YESNOCANCEL, "Report Request")
```

MB_ICONQUESTION là tên hằng cộng với tên hằng MB_YESNOCANCEL đưa ra kết quả là hai dấu ? và ba nút lệnh. title cũng xuất hiện nhờ vào giá trị thứ ba bên trong hàm MsgBox().

Lý do bạn gán các hàm MsgBox() vào biến là để có thể biết nút nào người dùng nhấn. Giả sử người dùng đã nhấn nút Yes trong Hình 9.6. Sau đó chương trình có thể in báo cáo. Tuy nhiên, nếu người dùng đã nhấn nút lệnh No, chương trình có thể mô tả những gì người dùng cần thực hiện để sẵn sàng cho việc in báo cáo (nạp giấy, bật máy in và ..v.v...). Nếu người dùng đã nhấn nút lệnh Cancel, chương trình sẽ biết rằng người dùng không muốn báo cáo tất cả.

Bảng 9.3 liệt kê các giá trị trả lại có thể có từ hàm MsgBox(). Nói cách khác, biến integer sẽ chứa một trong các giá trị của Bảng 9.3 sau khi mỗi hàm MsgBox() hoàn thành. Một dãy lệnh If lúc đó có thể kiểm tra để xem người dùng đã nhấn nút lệnh nào.

Bảng 9.3. *Các giá trị nút lệnh trả lại của hàm MsgBox()*

Giá trị	Giá trị CONSTANT.BAS	Diễn giải
1	IDOK	Người dùng đã nhấn nút OK
2	IDCANCEL	Người dùng đã nhấn nút Cancel
3	IDABORT	Người dùng đã nhấn nút Abort
4	IDRETRY	Người dùng đã nhấn nút Retry
5	IDIGNORE	Người dùng đã nhấn nút Ignore
6	IDYES	Người dùng đã nhấn nút Yes
7	IDNO	Người dùng đã nhấn nút No

Tóm tắt

Ví dụ 9.3 chỉ cách điều khiển hàm MsgBox() trước khi hiển thị 3 nút như trong Hình 9.6. Sau khi người dùng nhấp một nút, chương trình có thể sử dụng các lệnh If để xác định nút nào người dùng đã nhấn.

Ví dụ 9.3 *Hiển thị các hộp thông báo khác nhau.*

```
1: BPress = MsgBox("Are you ready for the report?",
MB_ICONQUESTION +
MB_YESNOCANCEL, "Report Request")
```

```
2: ' Check the button pressed
3: Select BPress
4: Case IDCANCEL: lblPress.Text = "You pressed Cancel"
5: Case IDYES: lblPress.Text = "You pressed Yes"
6: Case IDNO: lblPress.Text = "You pressed No"
7: End Select
```

Phân tích

Dòng 1 sẽ hiển thị hộp thông báo và cất nút nhấn trong biến có tên là Bpress. Dòng 3 bắt đầu một lệnh Select Case gán một trong ba chuỗi tới các nhãn trên mẫu biểu phù hợp với việc nhấn nút của người dùng. Bình thường, ví dụ này rất đơn giản; bạn sẽ thực hiện một trong nhiều loại thủ tục khác nhau, phụ thuộc vào người dùng đã nhấn nút lệnh nào trong ba nút lệnh.

Không cần viết mã lệnh Case Else bởi vì hộp thông báo ba nút có thể trả lại duy nhất một trong ba giá trị đã kiểm tra trong Ví dụ 9.2.

CÁC HÀM InputBox()

Khái niệm

Bạn sẽ thấy rằng hàm InputBox() thật dễ bởi vì nó hoạt động rất giống hàm MsgBox(). Có hai hàm InputBox(). Điểm khác nhau giữa chúng nằm ở chỗ kiểu dữ liệu mỗi hàm trả lại. Hàm InputBox() nhận các câu trả lời đầy đủ hơn hàm MsgBox() có thể nhận. Nhưng ngược lại MsgBox() trả lại một trong 7 giá trị cho biết việc nhấn nút lệnh của người dùng. Hàm InputBox() trả lại hoặc một chuỗi hoặc một giá trị dữ liệu variant mà người dùng đã nhập câu trả lời.

Có hai hàm InputBox(). Sau đây là các dạng thức hàm InputBox():

```
VariantVariable = InputBox( prompt [, [title] [, default][, xpos, ypos]]])
```

và

```
aStringVariable = InputBox$( prompt [, [title] [, default][, xpos, ypos]]])
```

Nét khác biệt giữa các hàm InputBox() nằm ở giá trị trả lại. Hàm InputBox() trả lại một kiểu giá trị dữ liệu variant và hàm InputBox$() (chú ý dấu $) trả lại một kiểu dữ liệu chuỗi. Thông thường, kiểu trả về

không quan trọng cho lắm. Hầu như bạn luôn luôn nhận trả lời InputBox() ở dạng một biến chuỗi, và nếu bạn sử dụng hàm InputBox(), Visual Basic sẽ chuyển các kiểu dữ liệu Variant thành các chuỗi khi cần.

Dấu nhắc (prompt) làm công việc giống như giá trị *msg* trong một hàm MsgBox(). Người dùng thấy dấu nhắc bên trong hộp nhận dữ liệu (input box) hiển thị trên màn hình. *Title* là tiêu đề trong thanh tiêu đề của hộp nhận dữ liệu. *default* là giá trị chuỗi mặc định mà Visual Basic sẽ hiển thị cho một câu trả lời mặc định, và người dùng có thể chấp nhận câu trả lời mặc định hoặc thay đổi nó.

Các vị trí *xpos* và *ypos* chỉ chính xác vị trí bạn muốn hộp nhận dữ liệu xuất hiện trên mẫu biểu. Giá trị *xpos* giữ số khoảng cách twip từ góc trái cửa sổ Form đến góc trái hộp nhận dữ liệu. Giá trị *ypos* giữ khoảng cách twip từ cạnh trên cùng của cửa sổ Form tới cạnh trên cùng của hộp nhận dữ liệu. Nếu bạn bỏ qua các giá trị *xpos* và *ypos*, Visual Basic canh giữa hộp nhận dữ liệu trên mẫu biểu.

Ghi chú

Các hộp nhận dữ liệu từ người dùng luôn có một nút lệnh OK và một nút lệnh Cancel. Nếu người dùng nhấn OK (hay nhấn phím Enter, và chọn OK là mặc định), câu trả lời trong hộp nhận dữ liệu được gởi tới biến đang được gán giá trị trả lại. Nếu người dùng nhấn nút Cancel, một chuỗi rỗng "" sẽ được trả về từ hàm InputBox().

Tóm tắt

Mã lệnh trong Ví dụ 9.3 sẽ hiển thị một hộp nhận dữ liệu để hỏi người dùng tên một công ty. Hoặc người dùng nhập vào một câu trả lời tại dấu nhắc hoặc nhấn nút lệnh Cancel để cho biết không có câu trả lời nào được thực hiện.

Ôn lại

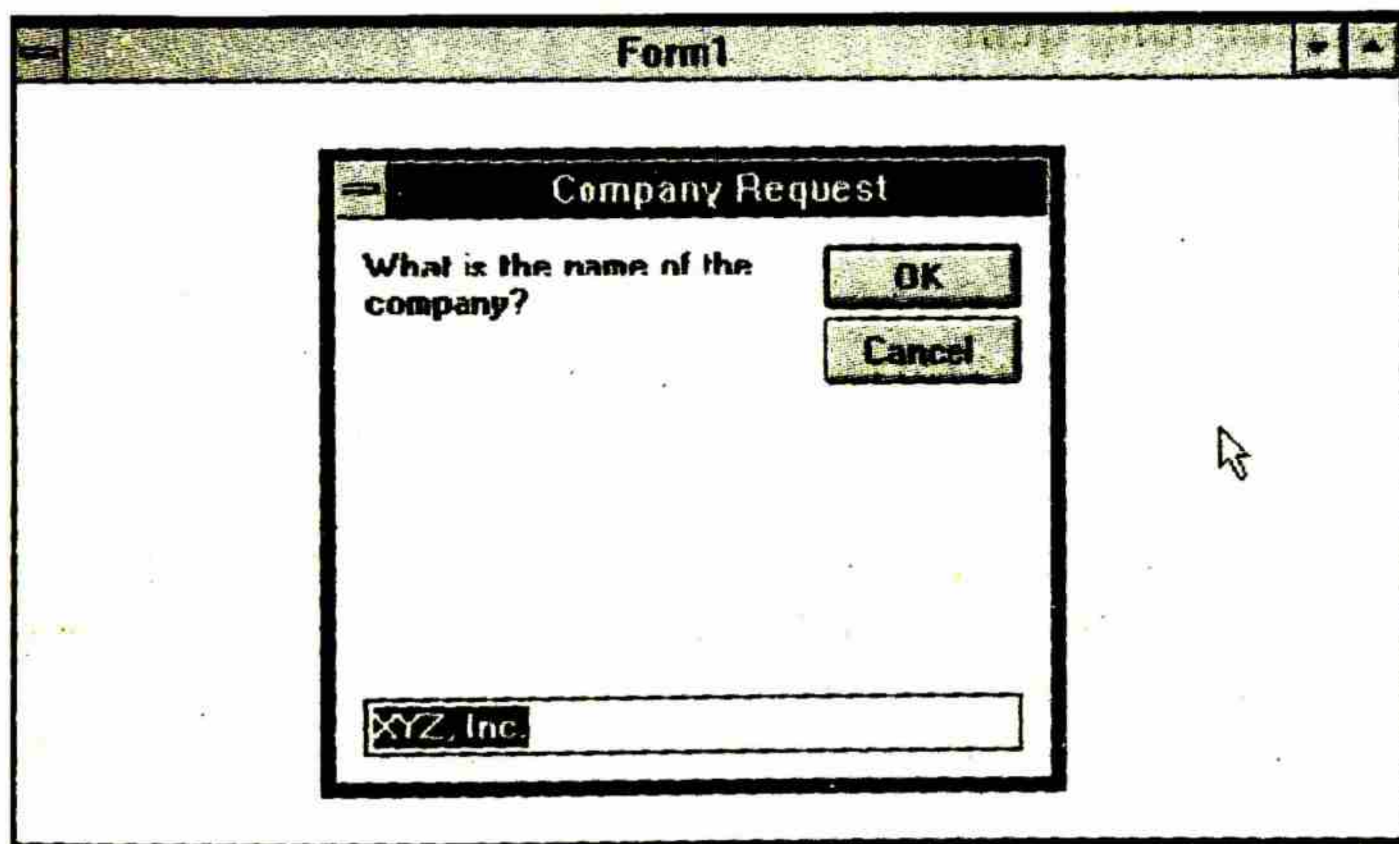
Các hàm InputBox() và InputBox$() nhận câu trả lời từ người dùng. Bạn có thể đề nghị một câu trả lời mặc định mà người dùng có thể chấp nhận hay thay đổi. Các hàm input box trả về câu trả lời vào một chuỗi hay một biến Variant mà bạn gán cho hàm.

Ví dụ 9.3. *Hộp nhận dữ liệu hỏi người dùng các câu hỏi.*

```
Dim CompName As String
CompName = InputBox$("What is the name of the company?",
"Company Request", "XYZ, Inc.")
If (CompName = "") Then
' The user pressed Cancel
Beep
MsgBox "Thanks anyway"
Else
' The user entered a company name
MsgBox "You entered " & CompName
End If
```

Kết quả

Hình 9.7 mô tả hộp thông báo được hiển thị từ Ví dụ 9.3.



Hình 9.7. *Các hộp nhận dữ liệu hỏi người dùng các câu hỏi và trả về những câu trả lời.*

Phân tích

Ví dụ 9.3 có thể là một phần của thủ tục biến cố sẽ thi hành khi chương trình sẵn sàng nhận một tên công ty. Dòng 1 định nghĩa một biến chuỗi vốn giữ câu trả lời của người dùng. Dòng 2 có chứa hàm InputBox$() hỏi tên công ty và hiển thị một câu trả lời mặc định (tự động hiện sáng như minh họa ở cuối Hình 9.7).

Ngay khi người dùng trả lời yêu cầu của hộp nhận dữ liệu, dòng 3 bắt đầu kiểm tra một trong hai kết quả sau: hoặc người dùng đã nhấn nút Cancel để trả lời hộp nhận dữ liệu, hoặc đã trả lời hộp nhận dữ liệu bằng việc nhấn nút OK. Nếu người dùng đã nhấn Cancel, hàm InputBox$() trả về một chuỗi null, và các dòng 5 và 6 sẽ phát tiếng bíp và hiển thị một hộp thông báo (message box) cảm ơn người dùng về sự cố gắng của họ. Dòng 8 và 9 trình bày lại tên công ty đã nhập bởi người dùng.

Bài tập

Kiến thức tổng quát

1. Ghi chú là gì?
2. Tại sao ghi chú lại quan trọng đến vậy?
3. Đúng hay Sai: Các ghi chú xuất hiện trên màn hình người dùng.
4. Đúng hay Sai: Các ghi chú xuất hiện trong cửa sổ Code.
5. Visual Basic hỗ trợ bao nhiêu loại ghi chú?
6. Tên bốn công dụng của các ghi chú.
7. Ai là người ghi chú?
8. Đúng hay Sai: Không có lý do gì để lo lắng về các ghi chú nếu bạn là lập trình viên duy nhất làm việc trên chương trình của bạn hiện nay và về sau.
9. Đúng hay Sai: Các hộp thông báo (message box) là những điều khiển.
10. Bạn điều khiển thế nào các nút lệnh xuất hiện trong hộp thông báo?
11. Bạn điều khiển thế nào văn bản xuất hiện trên các hộp thông báo?

12. Nghĩa của *modality* là gì?
13. Đúng hay Sai: Visual Basic bao gồm một lệnh message box và một hàm message box.
14. Đúng hay Sai: Visual Basic bao gồm một lệnh input box và một hàm input box.
15. Tại sao bạn ít sử dụng input box hơn là message box?
16. Các loại dữ liệu nào mà các hàm input box có thể trả về?
17. Bạn kiểm tra thế nào để bảo đảm người dùng đã nhấn Cancel trả lời một hộp nhận dữ liệu?
18. Giả sử bạn đã dùng King George, III như một giá trị mặc định trong một hàm input box. Người dùng phải làm gì để sử dụng giá trị mặc định đó?
19. Có bao nhiêu loại biểu tượng có thể hiển thị trong các hộp thông báo (message box)?
20. Có bao nhiêu loại biểu tượng có thể hiển thị trong các hộp nhận dữ liệu (input box)?
21. Bạn có thể điều khiển tiêu điểm (focus) nút lệnh như thế nào với các hộp thông báo?
22. Có bao nhiêu nút lệnh khác nhau có thể hiển thị trong các hộp thông báo?
23. Có bao nhiêu giá trị trả về trong các hàm MsgBox()?

Viết lệnh...

24. Nếu mã lệnh bạn viết nhận biết một tập tin trên đĩa riêng biệt bị hỏng, bạn muốn báo cho người dùng về lỗi nghiêm trọng này và muốn chắc chắn rằng người dùng không thể chuyển tới các chương trình Windows khác trước khi trả lời hộp thông báo lỗi của bạn. Kiểu đối số nào bạn sẽ dùng trong lệnh MsgBox?

Tìm lỗi kỹ thuật

25. Lập trình viên An Huy biết một tài liệu chương trình quan trọng như thế nào. Mặc dù cố gắng hết sức, An Huy đã gặp lỗi khi anh ấy cố thêm ghi chú sau tới cuối một dòng mã lệnh:

```
do until (endOfFile) Rem Continue until the end
```

Hãy giúp An Huy giải quyết vấn đề này.

Phần nâng cao

Viết thủ tục nút lệnh Click hỏi tuổi người dùng. Nếu người dùng nhấn Cancel, thì sẽ phát tiếng kêu bíp và hiển thị một hộp thông báo khác như đang nói chuyện. Nếu người dùng nhập vào một tuổi, dùng hàm Val() để đổi và lưu tuổi vào trong một biến số, sau đó hiển thị số năm cho đến khi ngừng không nhập nữa bằng cách lấy 65 trừ tuổi. *Gợi ý:* Hàm Str$() thực hiện ngược với hàm Val() và bạn có thể đổi số thành chuỗi và thêm vào hộp thông báo để hiển thị thông báo. (Chương 7 sẽ đề cập về Str$() với đầy đủ chi tiết.)

Bài 10

Vòng lặp

- Lệnh Do While Loop
- Lệnh Do Until Loop
- Các lệnh lặp Do khác
- Vòng lặp For

Bây giờ bạn thực sự sẵn sàng viết các chương trình mạnh mẽ! Bạn đã học một số điều khiển, đã định nghĩa một số biến, và đã viết các chương trình thực hiện quyết định. Khi máy tính thực hiện một công việc nào đó nhiều lần liên tục thì ta nói máy tính đang lặp.

Máy tính không thấy buồn chán. Sức mạnh cơ bản của máy tính là khả năng lặp một dãy các phép toán rất nhanh. Máy tính có thể xử lý các bảng cân đối khách hàng, tính trung bình lương của nhiều bộ phận, và hiển thị dữ liệu cho từng nhân viên công ty.

Bài này mô tả cách bạn có thể thêm vòng lặp vào các chương trình Visual Basic để chương trình có thể xử lý nhiều giá trị dữ liệu bằng việc sử dụng các lệnh lặp. Các lệnh lặp không những giúp bạn khi bạn có một số lớn dữ liệu cần xử lý, mà còn cho phép bạn sửa lỗi của người dùng và lặp các hàm chương trình xác định khi người dùng yêu cầu một lệnh lặp.

LỆNH Do While Loop

Khái niệm

Lệnh Do hỗ trợ nhiều dạng lặp khác nhau. Lệnh lặp Do While Loop có lẽ là lệnh lặp phổ biến nhất mà bạn sẽ thiết đặt trong các chương trình Visual Basic.

Khái niệm mới

Một khối (block) bao gồm một hay nhiều câu lệnh chương trình.

Lệnh Do While làm việc với các biểu thức quan hệ như lệnh If. Vì vậy, sáu toán tử quan hệ mà bạn đã đọc trong chương trước vẫn được chờ đợi ở đây. Tuy nhiên, ít có điều khiển nào chỉ thi hành một lần khối mã lệnh đơn lẻ, biểu thức quan hệ sẽ điều khiển các lệnh lặp.

Mã lệnh mà bạn đã thấy ở trước trong các thủ tục biến cố là mã lệnh tuần tự thi hành từng lệnh một, lệnh này tiếp lệnh khác theo thứ tự bạn đã nhập các lệnh. Vòng lặp sẽ thay đổi. Nhiều dòng trong chương trình vẫn thi hành tuần tự, nhưng một lệnh lặp sẽ tạo nên các khối mã lệnh để lặp một hay nhiều lần.

Giống câu lệnh If, cuối cùng kết thúc với lệnh End If, một lệnh lặp bao giờ cũng là một lệnh nhiều dòng kể cả lệnh bắt đầu và kết thúc vòng lặp. Sau đây là dạng của Do While Loop:

```
Do While (relational test)
    Block of one or more Visual Basic statements
Loop
```

Khái niệm mới

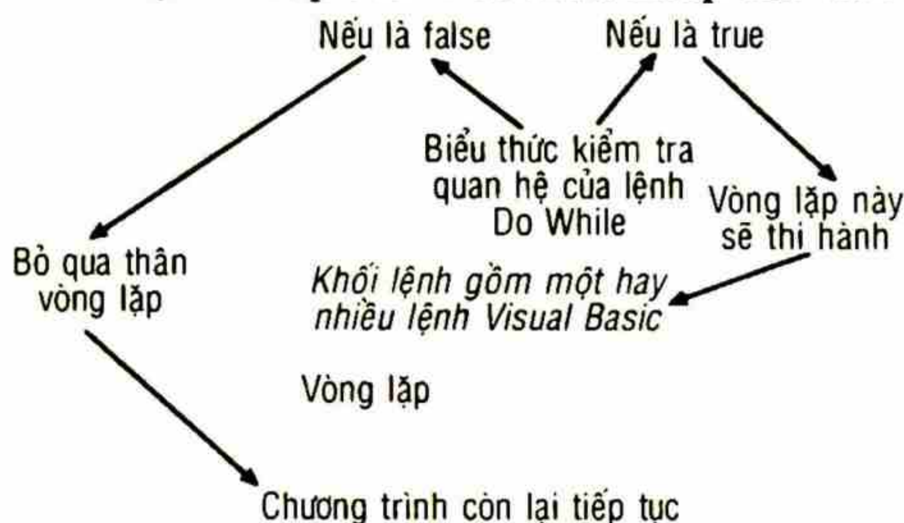
Một vòng lặp không xác định sẽ lặp mãi mãi.

Khối mã lệnh tiếp tục lặp khi biểu thức *kiểm tra quan hệ* (relational test) là true. Không có vấn đề gì lúc bạn chèn một hay nhiều mã lệnh trong khối. Tuy nhiên, khối lệnh sẽ thay đổi biến được sử dụng trong biểu thức kiểm tra quan hệ. Khối lệnh sẽ giữ việc lặp khi biểu thức kiểm tra quan hệ của lệnh Do While Loop vẫn là true. Biểu thức kiểm tra quan hệ phải là false bằng không chương trình của bạn sẽ nhập vào một vòng lặp vô tận và người dùng phải hủy bỏ việc thi hành chương trình theo một cách khác, chẳng hạn như nhấn tổ hợp phím Ctrl+Break.

Cảnh báo

Ngay cả khi bạn cung cấp một nút lệnh Exit như bạn đã từng sử dụng với các ứng dụng trong sách, chương trình vẫn luôn luôn bỏ qua việc nhấp nút lệnh Exit của người dùng nếu chương trình gặp một vòng lặp vô tận.

Hình 10.1 sẽ minh họa cách lệnh Do While Loop làm việc. Trong khi biểu thức kiểm tra quan hệ là true, khối lệnh trong thân vòng lặp sẽ tiếp tục thi hành. Khi biểu thức *relational test* là false, vòng lặp sẽ ngừng. Sau khi lệnh lặp ngừng, Visual Basic bắt đầu thi hành với câu lệnh tiếp theo sau lệnh Loop bởi vì từ khóa Loop cho biết kết thúc lệnh lặp.



Hình 10.1. *Hoạt động của lệnh Do While Loop trong khi biểu thức kiểm tra quan hệ là true.*

Ghi chú

Ngay khi biểu thức kiểm tra quan hệ trở thành false, lệnh lặp sẽ dừng và không thi hành thêm một lần nữa. Biểu thức kiểm tra quan hệ của lệnh Do While đặt ở đầu lệnh lặp. Vì vậy, nếu relational test *là false ngay lần đầu tiên vòng lặp bắt đầu, thì thân lệnh lặp sẽ không bao giờ thực hiện.*

Trong suốt cuốn sách này, bạn sẽ thấy các dòng thụt vào được sử dụng cho thân mã lệnh lặp. Bằng việc thụt sang phải phần thân lệnh lặp so với các câu lệnh dừng và lặp, bạn sẽ dễ dàng biết được đâu là nơi vòng lặp bắt đầu và kết thúc.

Tóm tắt

Ví dụ 10.1 chứa một thủ tục biến cố có một lệnh lặp Do While Loop hỏi tuổi người dùng. Nếu người dùng nhập vào một tuổi nhỏ hơn 10 hay lớn hơn 99, chương trình sẽ phát tiếng kêu bíp báo lỗi và hiển thị hộp nhận dữ liệu (input box) khác để hỏi tuổi. Chương trình tiếp tục việc hỏi tuổi khi người dùng nhập một tuổi ngoài vùng qui định.

Ôn lại

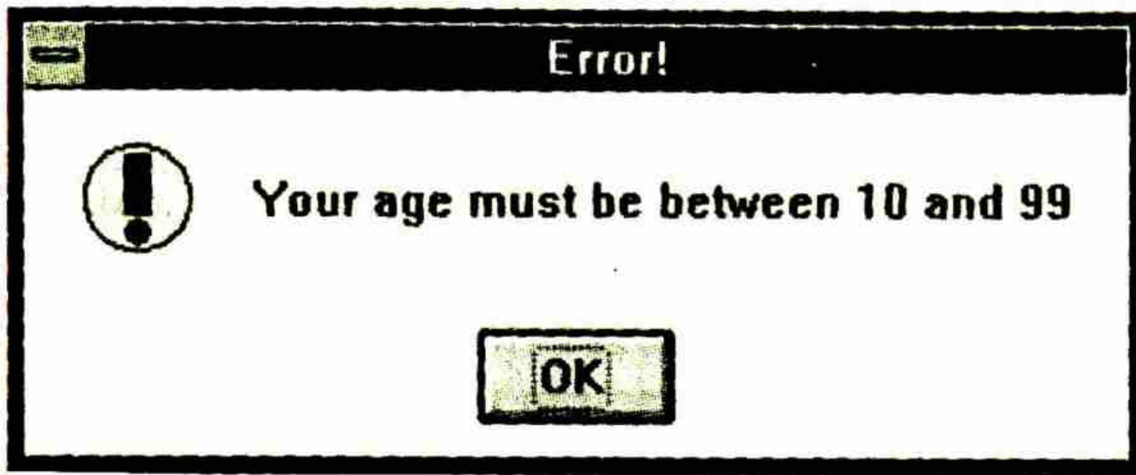
Lệnh Do While Loop sẽ tiếp tục thi hành một khối lệnh Visual Basic trong khi biểu thức kiểm tra quan hệ là true. Ngay khi biểu thức kiểm tra quan hệ trở thành false, vòng lặp sẽ dừng.

Ví dụ 10.1. *Lệnh Do While Loop sẽ thực hiện mãi trong khi giá trị biểu thức relational test là true.*

```
1: Dim StrAge As String
2: Dim Age As Integer
3: ' Get the age in a string variable
4: StrAge = InputBox$("How old are you?", "Age Ask")
5: ' Check for the Cancel command button
6: If (StrAge = "") Then
7: End
8: End If
9: ' Cancel was not pressed, so convert Age to integer
10: Age = Val(StrAge)
11: ' Loop if the age is not in the correct range
12: Do While ((Age < 10) Or (Age > 99))
13: ' The user's age is out of range
14: Beep
15: MsgBox "Your age must be between 10 and 99",
    MB_ICONEXCLAMATION, "Error!"
16: StrAge = InputBox("How old are you?", "Age Ask")
17: ' Check for the Cancel command button
18: If (StrAge = "") Then
19: End
20: End If
21: Age = Val(StrAge)
22: Loop
```

Kết quả

Hình 10.2 cho thấy hộp thông báo lỗi đã hiển thị ở dòng 15 nếu người dùng nhập vào một giá trị tuổi ít hơn 10 hay lớn hơn 99.



Hình 10.2. *Người dùng sẽ nhìn thấy hộp thông báo này khi tuổi vượt ngoài vùng giá trị.*

Phân tích

Dòng 1 và 2 định nghĩa hai biến, một biến chuỗi và một biến nguyên integer. Mã lệnh sẽ sử dụng biến chuỗi để nhận giá trị trả về từ hàm InputBox$(). Sử dụng biến chuỗi để bạn có thể kiểm tra nút Cancel bởi vì như bạn đã đọc trong chương trước, InputBox$() trả lại một chuỗi rỗng (null) nếu người dùng nhấn Cancel. Khi đó, mã lệnh sẽ dừng toàn bộ chương trình với một lệnh End (các dòng 7 và 19). Nếu người dùng nhập vào một tuổi (và không nhấn Cancel), mã lệnh chuyển chuỗi tuổi thành một số integer và kiểm tra để chắc chắn rằng tuổi ở trong phạm vi từ 10 đến 99.

Dòng 12 bắt đầu một lệnh lặp nếu tuổi nhỏ hơn 10 hoặc lớn hơn 99. Vòng lặp sẽ tiếp tục thi hành từ dòng 13 đến cuối vòng lặp ở dòng 22. Nếu tuổi ở ngoài vùng, thân vòng lặp sẽ thi hành. Dòng 14 sẽ phát tiếng bíp, như để gửi một tín hiệu tới người dùng rằng có gì đó không ổn. Dòng 15 sẽ hiển thị một hộp thông báo lỗi (như trong Hình 10.2) và sau khi người dùng nhấn OK trong hộp thông báo, người dùng lại được hỏi một lần nữa về tuổi với hàm InputBox$() trong dòng 16. Lúc đó vòng lặp kiểm tra để bảo đảm rằng người dùng đã không nhấn Cancel (dòng 17 đến dòng 20) và nếu không, mã lệnh sẽ đổi chuỗi tuổi thành một số nguyên integer và vòng lặp lại bắt đầu lần nữa. Nếu tuổi đã nhập vào trong phần trước của vòng lặp rơi vào trong vùng tuổi hợp lệ, chương trình sẽ hoàn thành (mã lệnh bất kỳ sau dòng 22 sẽ thực hiện). Ngược lại, vòng lặp bắt đầu lại một lần nữa.

Mã lệnh có một số lệnh thừa. Ví dụ, các dòng 4 và 16 có chứa hầu như cùng hàm InputBox$(), và việc kiểm tra thao tác nhấn nút lệnh Cancel sẽ xuất hiện hai lần trong chương trình. Có các câu lệnh lặp khác mà bạn sẽ học ở phần sau của chương; những lệnh này có thể giúp đơn giản hóa mã lệnh bằng cách gỡ bỏ một số lệnh dư thừa.

Có lẽ điều quan trọng nhất cần chú ý về Do While Loop trong Ví dụ 10.2 là thân của vòng lặp cung cấp một giá trị cho biểu thức kiểm tra quan hệ để dừng. Biểu thức kiểm tra quan hệ ở dòng 12 sẽ sử dụng biến Age mà thân của vòng lặp gán lại mỗi lần khối mã lệnh của vòng lặp thi hành. Vì vậy, giả sử người dùng nhập vào một giá trị tuổi khác, vòng lặp sẽ kiểm tra lại một bộ giá trị quan hệ khác ở dòng 12 và nó hy vọng biểu thức kiểm tra quan hệ sẽ sai (có nghĩa là tuổi nằm trong vùng giới hạn) và chương trình sẽ dừng vòng lặp. Nếu thân vòng lặp không làm gì với biến kiểm tra quan hệ, vòng lặp sẽ tiếp tục mãi mãi.

LỆNH Do Until Loop

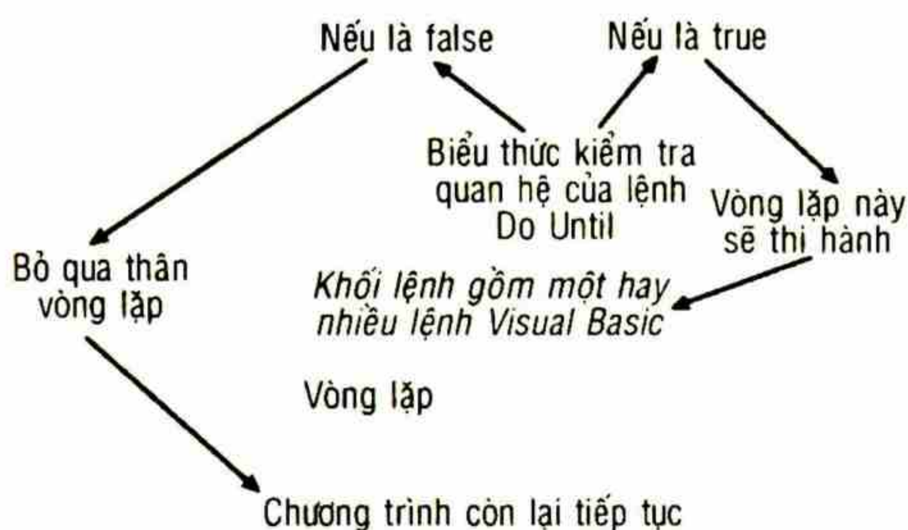
Khái niệm

Lệnh Do Whilte Loop sẽ tiếp tục thi hành thân vòng lặp khi biểu thức kiểm tra quan hệ là true, thì trái lại lệnh Do Until Loop sẽ thi hành thân vòng lặp khi biểu thức kiểm tra quan hệ là false. Logic của chương trình tại thời điểm lặp sẽ xác định loại vòng lặp nào làm việc tốt nhất trong trường hợp được cho.

Lệnh Do Until Loop làm việc hầu như chính xác giống lệnh Do While ngoại trừ việc vòng lặp Do Until tiếp tục thi hành phần thân vòng lặp cho đến khi biểu thức kiểm tra quan hệ là true. Giống Do While, lệnh Do Until là một lệnh lặp nhiều dòng có thể thi hành một khối mã lệnh gồm một hay nhiều dòng. Sau đây là dạng thức Do Until:

```
Do Until (relational test)
    Block of one ore more Visual Basic statements
Loop
```

Một lần nữa, hãy nhớ rằng biểu thức *relational test* phải là *false* để việc lặp tiếp tục. Hình 10.3 sẽ minh họa cách Do Until làm việc.



Hình 10.3. *Hoạt động của lệnh lặp Do Until tiếp tục trong khi biểu thức kiểm tra quan hệ là false.*

Tóm tắt

Bạn có thể sử dụng lệnh Do While hay Do Until cho đa số vòng lặp bất kỳ. Ví dụ 10.2 có chứa thủ tục biến cố kiểm tra tuổi có một vòng lặp Do Until. Vòng lặp đảm bảo rằng tuổi nằm giữa hai giá trị. Như bạn có thể thấy, biểu thức *relational test* cho Do Until ngược với biểu thức đã sử dụng trong Ví dụ 10.1 của vòng lặp Do While.

Ghi chú

Sử dụng vòng lặp làm cho biểu thức kiểm tra quan hệ trong sáng và rõ ràng. Đôi khi, logic tạo lệnh Do While lại rõ ràng hơn, tuy nhiên các vòng lặp khác dường như làm việc tốt hơn khi bạn khởi tạo chúng với lệnh lặp Do Until.

Ôn lại

Vòng lặp Do Until sẽ tiếp tục thi hành một khối lệnh Visual Basic khi biểu thức *relational test* là false. Ngay khi biểu thức *relational test* trở thành true (vòng lặp được hiểu là thực hiện một vòng lặp cho đến khi điều kiện trở thành *false*), vòng lặp sẽ dừng và chương trình tiếp tục trên dòng lệnh ngay sau lệnh Loop kết thúc vòng lặp.

Ví dụ 10.2. *Các vòng lặp Do Until cho đến khi biểu thức relational test trở thành true.*

```
1: Dim StrAge As String
2: Dim Age As Integer
3: ' Get the age in a string variable
4: StrAge = InputBox$("How old are you?", "Age Ask")
```

```
5: ' Check for the Cancel command button
6: If (StrAge = "") Then
7: End
8: End If
9: ' Cancel was not pressed, so convert Age to integer
10: Age = Val(StrAge)
11: ' Loop if the age is not in the correct range
12: Do Until ((Age >= 10) And (Age <= 99))
13: ' The user's age is out of range
14: Beep
15: MsgBox "Your age must be between 10 and 99",
    MB_ICONEXCLAMATION, "Error!"
16: StrAge = InputBox("How old are you?", "Age Ask")
17: ' Check for the Cancel command button
18: If (StrAge = "") Then
19: End
20: End If
21: Age = Val(StrAge)
22: Loop
```

Phân tích

Dòng 12 là dòng duy nhất đánh dấu sự khác nhau giữa Ví dụ 10.1 và 10.2. Bây giờ tuổi phải rơi vào vùng hợp lệ để vòng lặp có thể dừng.

Ghi chú

Không có ưu điểm nào thực sự trong việc sử dụng Do While hay Do Until. Việc sử dụng một trong hai lệnh này có lẽ bắt nguồn từ việc đáp ứng một ứng dụng đã cho bất kỳ.

CÁC LỆNH LẶP Do KHÁC

Khái niệm

Có một cặp vòng lặp Do khác làm việc y như hai vòng lặp đã đề cập ở phần trước. Đó là lệnh Do–Loop While và Do–Loop Until trông rất giống đồng nghiệp của chúng.

Tuy nhiên, hai dạng lặp mới sẽ kiểm tra các biểu thức *relational test* của chúng ở cuối vòng lặp thay vì ở đầu.

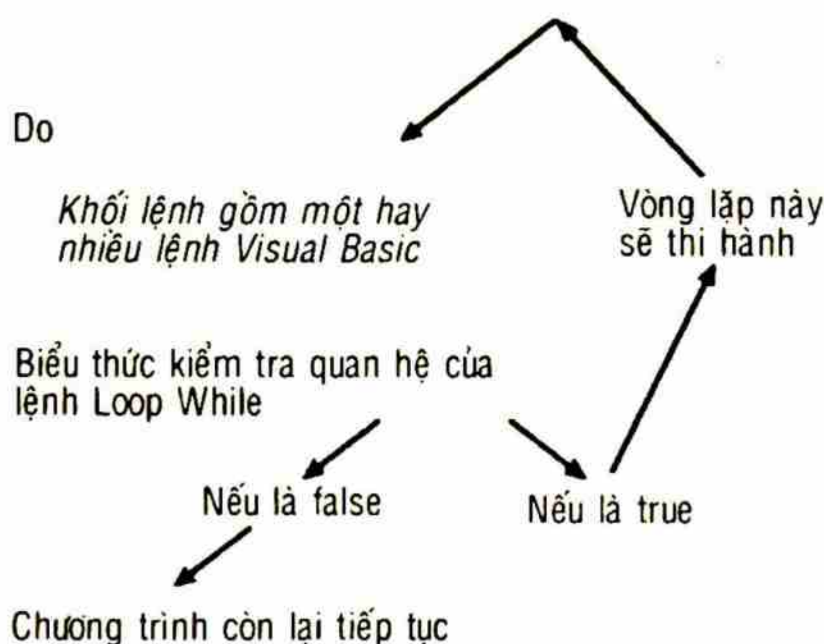
Nếu một vòng lặp bắt đầu với một lệnh Do đơn, vòng lặp sẽ kết thúc với lệnh Loop Whilet hoặc lệnh Loop Until. Sau đây là dạng thức Do–Loop While:

```
Do
    Block of one or more Visual Basic statements
Loop Until (relational test)
```

Lời nhắc

Điểm nổi bật trong Do–Loop While và Do–Loop Until là thực hiện thân vòng lặp trước lệnh Loop While.

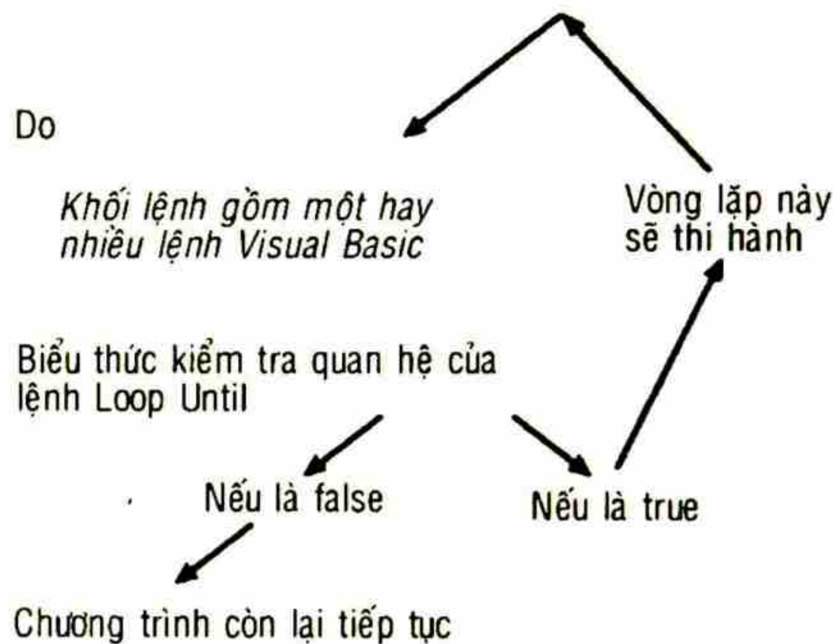
Lệnh Do trông nó cô đơn làm sao, có phải không? Hình 10.4 sẽ minh họa cách thức thi hành của vòng lặp Do–Loop While.



Hình 10.4. *Vòng lặp Do–Loop While không kiểm tra biểu thức kiểm tra quan hệ cho đến cuối thân vòng lặp.*

Hãy chú ý rằng, biểu thức kiểm tra quan hệ của vòng lặp Do–Loop While sẽ xuất hiện ở cuối vòng lặp thay vì đầu vòng lặp. Bạn sẽ sử dụng vòng lặp Do–Loop While khi muốn thân vòng lặp thi hành ít nhất một lần. Thông thường, bằng cách đặt biểu thức kiểm tra quan hệ ở cuối vòng lặp, bạn có thể bỏ qua mã lệnh dư thừa có thể được đòi hỏi nếu bạn sử dụng Do While.

Để hoàn thành các câu lệnh lặp, Visual Basic cũng hỗ trợ một lệnh Do–Loop Until. Giống Do–Loop While, lệnh Do–Loop Until sẽ kiểm tra biểu thức *relational test* ở cuối vòng lặp. Vì vậy, thân vòng lặp sẽ thi hành ít nhất một lần dẫu cho biểu thức *relational test* là gì đi nữa. Vòng lặp sẽ tiếp tục khi biểu thức *relational test* vẫn còn ở giá trị false. Hình 10.5 sẽ minh họa hoạt động của lệnh Do–Loop Until.



Hình 10.5. *Vòng lặp Do–Loop Until sẽ kiểm tra giá trị false của biểu thức kiểm tra quan hệ ở cuối thân vòng lặp.*

Tóm tắt

Ví dụ 10.3 có một thủ tục biến cố kiểm tra tuổi ngắn hơn nhiều so với các phiên bản trước. Biểu thức *relational test* sẽ xuất hiện ở cuối vòng lặp, cho nên việc thêm lệnh gọi hàm InputBox$() là không cần thiết.

Ôn lại

Biểu thức *relational test* sẽ xuất hiện ở cuối vòng lặp nếu bạn sử dụng lệnh lặp Do–Loop While. Thân vòng lặp luôn luôn thi hành ít nhất một lần. Thân vòng lặp sẽ thi hành nhiều hơn một lần khi biểu thức *relational test* duy trì giá trị true. Có một lệnh Do–Loop Until tương ứng chuyên kiểm tra một điều kiện false ở cuối thân vòng lặp.

Ví dụ 10.3. *Sử dụng lệnh Do–Loop While để kiểm tra quan hệ ở cuối vòng lặp.*

```
1: Dim StrAge As String
2: Dim Age As Integer
3: Do
4: StrAge = InputBox("How old are you?", "Age Ask")
5: ' Check for the Cancel command button
6: If (StrAge = "") Then
7: End
8: End If
9: Age = Val(StrAge)
10: If ((Age < 10) Or (Age > 99)) Then
11: ' The user's age is out of range
12: Beep
13: MsgBox "Your age must be between 10 and 99",
    MB_ICONEXCLAMATION, "Error!"
14: End If
15: Loop While ((Age < 10) Or (Age > 99))
```

Phân tích

Vòng lặp bắt đầu ngay ở dòng 3. Thân vòng lặp sẽ thi hành ít nhất một lần, cho nên hàm InputBox$() sẽ xuất hiện ngay trong vòng lặp. Bằng cách đặt hàm InputBox$() bên trong vòng lặp, bạn bỏ qua nhu cầu đặt hàm này vào mã lệnh hai lần (một lần trước vòng lặp và một lần trong vòng lặp, như khi sử dụng các lệnh lặp trước trong Ví dụ 10.1 và 10.2).

Dòng 10 phải kiểm tra để bảo đảm rằng giá trị InputBox$() ở trong hay ở ngoài vùng tuổi để thông báo lỗi có thể hiển thị.

Ghi chú

Trong ứng dụng đơn giản các câu lệnh lặp ở đây, lệnh lặp Do–While Loop đòi hỏi ít mã lệnh hơn các lệnh lặp Do While và Do Until. Bằng cách thay đổi biểu thức relational test ở dòng 15, *một lệnh Do Until cũng sẽ làm việc. Hai lệnh lặp cuối này sẽ không giảm mã lệnh, trong từng trường hợp. Logic của ứng dụng sẽ xác định vòng lặp nào làm việc tốt nhất.*

VÒNG LẶP For

Khái niệm

Vòng lặp For (đôi khi còn gọi là vòng lặp For–Next) cũng tạo một vòng lặp. Tuy nhiên, không giống các vòng lặp Do, vòng lặp For sẽ lặp lại một số lần xác định. Dạng thức vòng lặp For trông dễ nản chí hơn các vòng lặp Do, nhưng sau khi bạn làm chủ các lệnh Do, bạn sẽ gặp một vài vấn đề thi hành các vòng lặp For khi mã lệnh của bạn cần lặp một nhóm mã lệnh với số lần xác định.

Không có một vòng lặp tối ưu để sử dụng trong tất cả trường hợp. Lệnh For sẽ cung cấp một cơ chế xây dựng vòng lặp thứ 5 trong Visual Basic. Một vòng lặp For luôn luôn bắt đầu với lệnh For và kết thúc với lệnh Next. Sau đây là dạng thức vòng lặp For:

```
For CounterVar = StartVal To EndVal [Step IncrementVal]
    Block of one or more Visual Basic statements
Next CounterVar
```

Vòng lặp trong Ví dụ 10.4 sẽ tính tổng các số từ 1 đến 10.

Ví dụ 10.4. *Cộng các số từ 1 đến 10.*

```
1: Sum = 0
2: For Number = 1 To 10
3: Sum = Sum + Number
4: Next Number
```

Number là một biến đếm (counter Var) ở dạng vòng lặp For (dòng 2). Biến đếm phải là một biến. 1 là giá trị bắt đầu (StartVal) (dòng 2). Giá trị bắt đầu có thể là một số, một biểu thức hay một biến. 10 là giá trị kết thúc (EndVal) (vẫn trong dòng 2). Giá trị kết thúc có thể là một số, một biểu thức, hay một biến. Bước nhảy Step không được xác định ở đây. Trong dạng thức câu lệnh For, Step IncrementVal là một tùy chọn. Nếu bạn không xác định một giá trị Step, Visual Basic sẽ giả sử giá trị Step là 1. Vì vậy, cả hai lệnh For sau là chính xác như nhau:

```
For Number = 1 To 10
```

Và

```
For Number = 1 To 10 Step 1
```

Vòng lặp For tính tổng của Ví dụ 10.4 gán giá trị ban đầu tới CounterVar và StartVal trong dòng 2. Vì vậy, Number được gán là 1 ở đầu vòng lặp. Lúc đó, Visual Basic sẽ thi hành thân vòng lặp giá trị 1 cho Number. Với Number bằng 1, dòng 3 làm việc như sau trong lần đầu tiên của vòng lặp:

```
Sum = Sum + 1
```

Khi Visual Basic thi hành lệnh Next Number, Visual Basic trở lại đầu vòng lặp (lệnh For), cộng giá trị Step = 1 vào Number, và tiếp tục lặp lại bằng cách sử dụng giá trị 2 của Number trong thân vòng lặp. Vì vậy, lần thứ hai qua vòng lặp, dòng 3 sẽ làm việc như sau:

```
Sum = Sum + 2
```

Vòng lặp sẽ tiếp tục việc cộng giá trị Step mặc định là 1 với Number mỗi lần vòng lặp thi hành. Khi Number đạt giá trị 10 (EndVal), vòng lặp sẽ hoàn thành và lệnh sau lệnh Next sẽ tiếp tục.

Lời nhắc

Hãy nhớ, vòng lặp For sẽ dừng khi biến đếm CounterVar *có giá trị lớn hơn giá trị* EndVal. *Có một ngoại lệ với điều này: nếu bạn viết một giá trị Step âm, vòng lặp sẽ dừng khi* CounterVar *đạt giá trị nhỏ hơn* EndVal, *như bạn sẽ thấy một lát nữa trong phần này.*

Bạn không cần lệnh For để tính tổng các giá trị từ 1 đến 10. Bạn có thể viết một lệnh gán dài giống như sau:

```
Sum = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10
```

Bạn cũng có thể viết lại mã lệnh thành các câu lệnh gán như sau:

```
Sum = Sum + 1
Sum = Sum + 2
Sum = Sum + 3
Sum = Sum + 4
Sum = Sum + 5
Sum = Sum + 6
Sum = Sum + 7
Sum = Sum + 8
Sum = Sum + 9
Sum = Sum + 10
```

Không những phương pháp này vô cùng khó khăn, mà còn gay cấn ở chỗ những gì bạn cần để cộng 100 số integer đầu tiên? Ví dụ 10.5 sẽ minh họa.

Ví dụ 10.5. *Cộng các số từ 1 tới 100.*

```
Sum = 0
For Number = 1 To 100 ' Only this line changes
Sum = Sum + Number
Next Number
```

Vòng lặp sau sẽ hiển thị 5 hộp thông báo (message box):

```
For c = 1 To 20 Step 4
    MsgBox "This is a message box"
Next c
```

Khái niệm mới

loop iteration ***là sự lặp vòng.***

Vòng lặp đếm lên từ 1 đến 20 với bước nhảy là 4, sẽ đặt từng giá trị đếm vào biến tên c và in một hộp thông báo mỗi lần. Giá trị Step sẽ thay đổi cách Visual Basic cập nhật CounterVar mỗi lần lặp đầy đủ.

Nếu bạn xác định một giá trị Step âm, Visual Basic sẽ đếm giảm dần. Vòng lặp sau sẽ phát tiếng bíp từ loa PC 5 lần:

```
For i = 5 To 1 Step -1
    Beep
Next i
```

Cảnh báo

Nếu bạn xác định một giá trị Step âm, EndVal *phải nhỏ hơn* StartVal *hay Visual Basic sẽ thi hành vòng lặp một lần duy nhất.*

Bạn có thể dừng các vòng lặp sớm: Đôi khi, bạn xử lý dữ liệu vào của người dùng hoặc nhiều giá trị dữ liệu đang sử dụng trong các câu lệnh lặp, và một ngoại lệ sẽ xảy ra trong dữ liệu đòi hỏi dừng lặp ngay lập tức. Ví dụ, bạn chọn các giá trị lương cho 10 bộ phận trong một công ty bên trong vòng lặp For chia làm 10 lần. Tuy nhiên, người dùng có thể nhập vào số 0 cho giá trị lương của một bộ phận, để chỉ ra rằng không có dữ liệu lương cho bộ phận đó. Thay vì hoàn thành việc lặp,

chương trình của bạn có lẽ cần thoát vòng lặp tại điểm đó bởi vì thông tin báo cáo bộ phận đầy đủ không được tập hợp lúc đó.

Lệnh Exit Do và Exit For sẽ tự dừng các vòng lặp. Không quan tâm tới kết quả kiểm tra quan hệ của vòng lặp Do hay số lần lặp còn lại trong biến đếm ở bên trái vòng lặp For, khi Visual Basic gặp một lệnh Exit Do hay Exit For, Visual Basic sẽ thoát vòng lặp ngay lập tức và chuyển việc thi hành xuống câu lệnh ngay sau vòng lặp.

Điển hình, câu lệnh If sẽ kích hoạt một trong các lệnh Exit giống như sau:

```
For Divisions = 1 To 10
' Code to get a sales value
If (sales = 0) Then
Exit For ' Quit the loop early
End If
' Process the rest of the code
Next Divisions
```

Lệnh If đảm bảo rằng lệnh Exit For sẽ thi hành chỉ dưới một điều kiện xác định (một giá trị lương thiếu). Không có lệnh Exit For kích hoạt điều kiện xác định, vòng lặp sẽ thực hiện bình thường.

Tóm tắt

Ví dụ 10.6 bao hàm toàn diện vòng lặp For tính lãi kép cho giá trị đầu tư ban đầu của $1,000.00. Mã lệnh sẽ xuất hiện bên trong thủ tục biến cố Click cho một nút lệnh có tên là Cmdlnt. Trong trường hợp bạn không quen với lãi kép, mỗi năm tổng số tiền đã đầu tư, kể cả tiền đầu tư trước đó, phát sinh thêm tiền lãi. Mỗi chu kỳ, thông thường là 1 năm, nghĩa là tiền đầu tư của năm khác phải được cộng với giá trị của tiền đầu tư. Một vòng lặp For là hoàn hảo cho việc tính toán đầu tư. Ví dụ 10.6 sẽ sử dụng 5 vòng tính lãi.

Ôn lại

Vòng lặp For sẽ lặp một khối gồm một hay nhiều mã lệnh của Visual Basic. Không giống các vòng lặp Do, vòng lặp For lặp lại một số lần xác định được điều khiển bởi các giá trị điều khiển của lệnh For và biến.

Ví dụ 10.6. *Dùng một vòng lặp For để tính lãi kép.*

```
1: Sub cmdInt_Click ()
2: ' Use a For loop to calculate a final total
3: ' investment using compound interest.
4: ' Num is a loop control variable
5: ' IRate is the annual interest rate
6: ' Term is the Number of years in the investment
7: ' InitInv is the investor's initial investment
8: ' Interest is the total interest paid
9: Dim IRate, Interest As Single
10: Dim Term, Num As Integer
11: Dim InitInv As Currency
12:
13: IRate = .08
14: Term = 5
15: InitInv = 1000.00
16: Interest = 1 ' Begin at one for first compound
17:
18: ' Use loop to calculate total compound amount
19: For Num = 1 To Term
20: Interest = Interest * (1 + IRate)
21: Next
22:
23: ' Now we have total interest,
24: ' calculate the total investment
25: ' at the end of N years
26: lblFinalInv.Caption = InitInv * Interest
27: End Sub
```

Phân tích

Phân tích này tập trung vào vòng lặp và không quan tâm tới việc tính toán. Điều quan trọng nhất mà bạn có thể thực hiện lúc này là làm chủ lệnh lặp For. Các dòng 1 đến 8 chứa những ghi chú khá bao quát. Ghi chú chứa nội dung mô tả biến để một ai đó nhìn mã lệnh hay thay đổi mã lệnh sau này sẽ biết biến chứa những gì.

Sau khi chương trình định nghĩa tất cả các biến từ dòng 9 đến dòng 11, biến được khởi tạo với những giá trị ban đầu từ dòng 13 đến dòng 16. Nếu bạn sử dụng thủ tục biến cố này, hãy chắc chắn để thêm một nhãn (label) có tên lblFinallnv vào một mẫu biểu và thêm một nút lệnh vào mẫu biểu có tên cmdlnt. Dòng 15 sẽ gây rắc rối cho bạn khi nhập liệu trừ khi bạn nhớ các ký tự hậu tố của dữ liệu đã mô tả trong Chương 2. Visual Basic sẽ sử dụng dấu pound, #, để chỉ các giá trị chính xác kép, và Visual Basic sẽ giả sử rằng 1000.00 là một giá trị chính xác kép (không hiểu tại sao) và sẽ đổi 1000.00 thành 1000# ngay sau khi bạn nhấn Enter ở cuối dòng! Đừng lo lắng về sự chọn lựa của Visual Basic ở đây.

Phần quan trọng nhất của chương trình này sẽ xuất hiện trên các dòng 19 đến dòng 21. Dòng 19 bắt đầu một vòng lặp For lặp qua từng chu kỳ tỉ giá đầu tư (5 lần), lãi kép trên số tiền đầu tư theo ngày trên dòng 20. Một lần nữa, đừng để việc quản lý tài chính làm bận tâm bạn. Việc tính toán ít quan trọng hơn việc tìm hiểu quá trình lặp. Sau khi vòng lặp hoàn thành, dòng 26 hoàn tất thủ tục biến cố bằng việc đặt lãi kép vào nhãn (label). Bằng cách đó, nếu bạn thi hành thủ tục biến cố này trong ứng dụng riêng của bạn, số tiền đầu tư sẽ xuất hiện trong cửa sổ như một số chính xác đơn với nhiều vị trí thập phân. Bạn sẽ học cách định dạng dữ liệu thành dollar và cent trong Chương 7.

Bài tập

Kiến thức tổng quát

1. Vòng lặp là gì?
2. Visual Basic hỗ trợ bao nhiêu loại lệnh lặp?
3. Visual Basic hỗ trợ bao nhiêu lệnh Do khác nhau?
4. Đúng hay Sai: Một khối có thể chứa một lệnh đơn.
5. Vòng lặp không xác định là gì?
6. Bạn có thể tận dụng một vòng lặp để sửa chữa lỗi do người dùng gây ra như thế nào?
7. Có bao nhiêu lần vòng lặp sau sẽ thi hành?

```
I = 10
Do While I > 1
I = I – 1
Loop
```

8. Có bao nhiêu lần vòng lặp sau sẽ thi hành?

```
I = 10
Do While I >= 1
I = I – 1
Loop
```

9. Có bao nhiêu lần vòng lặp sau sẽ thi hành?

```
I = 10
Do Until I > 1
I = I – 1
Loop
```

10. Có bao nhiêu lần vòng lặp sau sẽ thi hành?

```
For I = 1 To 10
Beep
Loop
```

11. Visual Basic sẽ hiểu giá trị Step là bao nhiêu nếu bạn không xác định giá trị Step nào?
12. Lệnh nào, Do hay For, hỗ trợ một vòng lặp thường tiếp tục với một số lần xác định?
13. Lệnh nào, Do hay For, hỗ trợ một vòng lặp thường tiếp tục tùy theo biểu thức kiểm tra quan hệ?
14. Điểm khác nhau giữa vòng lặp Do While và Do Until là gì?
15. Điểm khác nhau giữa vòng lặp Do While và Do–Loop While là gì?
16. *Iteration* là gì?
17. Bạn có thể ép một vòng lặp For đếm xuống thay vì đếm lên như thế nào?
18. Nếu một giá trị khởi tạo ban đầu của vòng lặp For lớn hơn giá trị kết thúc, điều gì đúng cho giá trị tăng thêm?
19. Câu lệnh nào dừng các vòng lặp Do và For sớm?

Lập trình...

20. Viết chương trình gán giá trị 34 cho một biến và sau đó hỏi người dùng đoán số đang sử dụng trong một hộp nhận dữ liệu (input box).

Tìm lỗi kỹ thuật

21. Lập trình viên An Huy vì thiếu kinh nghiệm đã viết một vòng lặp For sau y như một vòng lặp vô định vậy. Bạn có thể nhận ra vấn đề không?

```
For i = 1 To 25
Total = Total * I
i = i – 1
Next I
```

22. An Huy muốn vòng lặp For của anh ấy lặp đi lặp lại 100 lần nhưng anh ấy đang có vấn đề. Hãy cho An Huy biết có gì sai trong dòng lệnh sau:

```
For I = 100 To 1 Step 1
```

Phần nâng cao

Vòng lặp lồng (nested loop) là một vòng lặp trong một vòng lặp. Vòng lặp ở ngoài xác định có bao nhiêu lần vòng lặp sẽ thi hành. Hãy xem liệu bạn có thể xác định có bao nhiêu lần mã lệnh sau phát tiếng bíp tới người dùng không.

```
For i = 1 To 5 ' The outer loop
For j = 1 To 3 ' The inner loop
Beep
Next j ' Inner loop completes before
Next i ' outer loop iterates again
```

Bài thực hành 5

Lợi ích của điều khiển

Tóm tắt

Chương này chỉ cho bạn cách cung cấp tài liệu và điều khiển chương trình. Khi bạn tăng thêm khả năng trình ứng dụng Visual Basic, chúng trở nên phức tạp hơn. Và lúc đó, quá trình bảo trì chương trình thêm khó khăn. Cần lưu ý đến thông tin tóm tắt tài liệu trợ giúp về chương trình và về các dòng lệnh phức tạp.

Bây giờ bạn có thể bổ sung điều khiển vào chương trình bằng cách sử dụng các vòng lệnh lặp, để lặp lại một phần chương trình. Máy tính là công cụ tuyệt vời cho phép tính lặp nhiều lần trước khi đưa ra kết quả. Vòng lặp cũng cung cấp cho bạn cách kiểm tra dữ liệu nhập vào của người dùng là đúng hay sai và lặp lại dấu nhắc nhập liệu để người dùng sử dụng đúng dạng thức và giá trị trong vùng được yêu cầu.

Trong chương này, bạn đã nắm bắt:

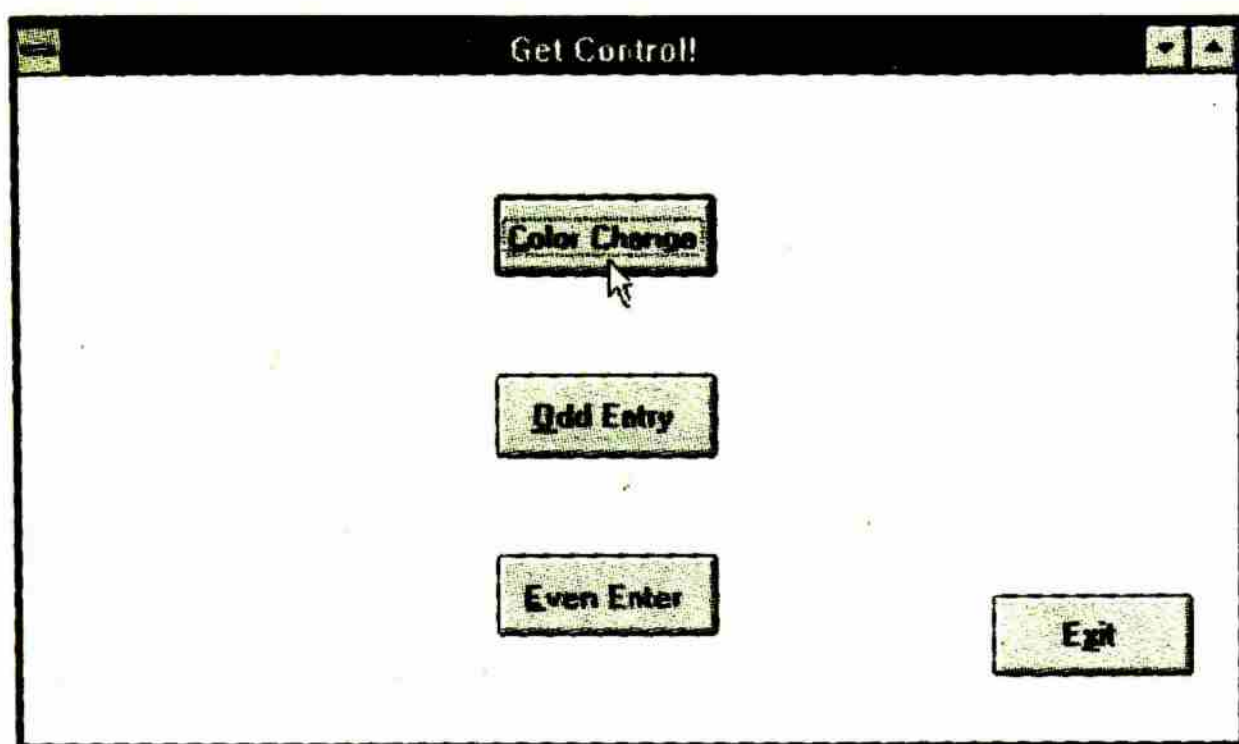
- Cách chú thích mã lệnh làm tài liệu trợ giúp
- Khi nào hộp thông báo giúp đỡ người dùng nhiều hơn điều khiển nhãn
- Cách nhận kết quả phản hồi từ các hộp nhận dữ liệu
- Sức mạnh của vòng lặp bổ sung trong chương trình

Mô tả chương trình

Hình P5.1 minh họa ứng dụng PROJECT5.VBP lúc nạp và chạy chương trình. Mẫu biểu project có 4 nút lệnh, thực hiện những nhiệm vụ khác nhau được minh họa qua những đề mục khác nhau.

Ba nút lệnh ở giữa thực hiện các hành động sau đây:

- Nút lệnh Color Change thay đổi nhanh màu nền mẫu biểu qua một số màu.
- Nút lệnh Odd Entry yêu cầu người dùng nhập một số lẻ.
- Nút lệnh Even Entry yêu cầu người dùng nhập một số chẵn.



Hình P5.1. *Màn hình đang mở của project.*

Hoạt động chương trình

Nhấp nút lệnh Color Change. Màu nền mẫu biểu sẽ đổi màu rất nhanh trong khi phát tiếng bíp. Các màu thay đổi nhanh như thế khiến bạn không thể xem tất cả các màu.

Tiếng bíp được thực hiện nhờ 4 lệnh Beep để phát tiếng ra loa PC 4 lần giữa lúc thay đổi màu. Mục đích thật của tiếng bíp là làm chậm lại quá trình thay đổi màu. Trong phần lớn máy tính, các màu sẽ thay đổi quá nhanh nếu không có tiếng bíp trung gian, và bạn không thể thấy bất kỳ điều gì ngoại trừ một cảnh mờ. Ngay khi dùng tiếng bíp, quá trình thay đổi màu cũng nhanh hơn khả năng nhìn thấy của mắt bạn.

Nút lệnh Odd Entry yêu cầu người dùng một số lẻ, bằng cách dùng hộp nhập được minh họa trong Hình P5.2. Thủ tục biến cố Click của nút lệnh duy trì quá trình lặp cho đến khi người dùng nhập vào một số lẻ.



Hình P5.2. *Chương trình yêu cầu một số lẻ.*

Nút lệnh thứ ba, Even Entry, yêu cầu một số chẵn và duy trì vòng lặp cho đến khi người dùng nhập số chẵn.

Thủ tục thay đổi màu

Ví dụ P5.1 trình bày thủ tục biến cố Click của nút lệnh Color Change.

```
1: Sub cmdColor_Click ()
2: ' Wildly Change the color of the form
3: Dim ColorVal As Integer
4: For ColorVal = 0 To 15 ' Step through the color values
5: frmControl.BackColor = QBColor(ColorVal)
6: Beep ' These beeps simply add
7: Beep ' to the intensity of the
8: Beep ' color change and slow
9: Beep ' things down some
10: Next ColorVal
11: End Sub
```

Mô tả

1: Nút lệnh tên cmdColor, cho nên tên thủ tục biến cố Click là cmdColor_Click().

2: Chú thích giải thích mục đích thủ tục biến cố.

3: Biến số nguyên lưu giá trị thay đổi màu vòng lặp For.

4: Dòng này bắt đầu vòng lặp For lặp lại 16 lần, bằng quá trình gán các số từ 0 đến 15 cho biến ColorVar.

5: Hàm QBColor() có sẵn giúp bạn thay đổi dễ dàng các giá trị màu. Hàm có sẵn QBColor() đòi hỏi giá trị số trong cặp ngoặc đơn từ 0 tới 15. Mỗi số đại diện cho một giá trị màu khác nhau. Sử dụng QBColor() sẽ dễ hơn xác định một giá trị thập lục phân.

6: Lệnh Beep làm chậm quá trình thay đổi màu và thêm dáng vẻ cho chương trình.

7: Lệnh Beep làm chậm quá trình thay đổi màu và thêm dáng vẻ cho chương trình.

8: Lệnh Beep làm chậm quá trình thay đổi màu và thêm dáng vẻ cho chương trình.

9: Lệnh Beep làm chậm quá trình thay đổi màu và thêm dáng vẻ cho chương trình.

10: Dòng này kết thúc vòng lặp For.

11: Dòng này kết thúc thủ tục biến cố.

Lấy giá trị lẻ

Khi người dùng nhấp nút Odd Entry, chương trình sẽ yêu cầu một số lẻ, bằng cách dùng hàm InputBox$(). Hàm InputBox$() đợi một giá trị chuỗi. Nếu người dùng nhấp nút Cancel, cửa sổ Form sẽ được trả lại. Ngược lại, thủ tục biến cố sẽ chờ đợi một số lẻ và duy trì yêu cầu nhập số với vòng lặp Do cho đến khi người dùng nhập vào một số lẻ.

Ví dụ P5.2. *Thủ tục biến cố yêu cầu một số lẻ.*

```
Sub cmdOdd_Click ()
' Request an odd number
Dim OddStr As String
Dim OddNum As Integer
Do
OddStr = InputBox$("Enter an odd number", "Get Odd")
If (OddStr = "") Then ' Quit if pressed Cancel
Exit Do ' Quits the Do loop early
End If
OddNum = Val(OddStr) / 2
```

```
11: ' The integer OddNum holds the exact value
12: ' of the Val(OddStr) / 2 if OddNum is even
13: Loop Until (OddNum <> (Val(OddStr) / 2))
14: End Sub
```

Mô tả

1: Thuộc tính Name của nút lệnh có giá trị cmdOdd, cho nên tên của thủ tục biến cố Click là cmdOdd_Click().

2: Chú thích giải thích mục đích của thủ tục.

3: Dòng này định nghĩa một biến chuỗi lưu kết quả hàm InputBox$().

4: Dòng này định nghĩa một biến số nguyên lưu kết quả hàm InputBox$().

5: Dòng này bắt đầu vòng lặp.

6: Chương trình đợi người dùng nhập một số lẻ, và nó hiển thị đề mục hộp nhập thích hợp. Dùng hàm InputBox$() để nhận chuỗi từ người dùng.

7: Nếu người dùng nhấn nút lệnh Cancel, chương trình sẽ thoát vòng lặp.

8: Dòng lệnh này sẽ kết thúc vòng lặp Do.

9: Dòng lệnh này kết thúc lệnh If.

10: Dòng lệnh này chuyển chuỗi nhập vào thành số, chia số đó cho 2, và lưu kết quả trong biến số nguyên.

11: Chú thích nhiều dòng mô tả quá trình kiểm tra số lẻ.

12: Tiếp tục chú thích ở dòng trên.

13: OddNum lưu giá trị số nguyên được tính trong dòng 10. Nếu biến nguyên OddNum có giá trị thập phân từ kết quả chia giá trị do người dùng nhập cho 2, người dùng đã nhập một số lẻ, và chương trình cần lặp lại và hỏi số lẻ khác.

14: Dòng này kết thúc thủ tục biến cố.

Ghi chú

Thủ tục biến cố cmdEven_Click() tương tự cmdOdd_Click() ngoại trừ việc nó kiểm tra số chẵn thay vì số lẻ.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi chương trình và Visual Basic. Chương kế tiếp sẽ giải thích thêm các lệnh Visual Basic và mô tả cách bổ sung các điều khiển mới vào mẫu biểu của bạn và hiển thị số lượng dữ liệu lớn.

Chương VI

Bài 11

Mảng và danh sách

- Các mảng giữ dữ liệu
- Ưu điểm khi sử dụng mảng
- Hộp danh sách: điều khiển làm việc giống mảng
- Hộp combo

Biến và điều khiển lưu trữ dữ liệu mà chương trình Visual Basic xử lý. Bây giờ bạn đã thấy các kiểu dữ liệu biến khác nhau mà Visual Basic sẽ hỗ trợ, cùng với ba điều khiển căn bản được sử dụng trên các mẫu biểu: *nút lệnh* (command button), *nhãn* (label), và *hộp nhập* (Text Box).

Bài này thảo luận chi tiết hơn về các biến và điều khiển bằng cách chỉ cho bạn cách quản lý việc lưu trữ dữ liệu biến mới. Bằng cách lưu biến vào mảng, bạn sẽ có thể xử lý số lượng dữ liệu lớn hơn với ít mã lệnh hơn.

Sau khi học về khái niệm mảng và cách truy xuất các biến mảng, bạn sẽ tìm hiểu thêm về những điều khiển có thể đặt trên mẫu biểu. Các điều khiển này, là *hộp danh sách* (List Box) và *hộp combo* (Combo Box), giúp bạn hiển thị mảng dữ liệu trên mẫu biểu một cách đơn giản.

CÁC MẢNG GIỮ DỮ LIỆU

Khái niệm

Mảng là một bảng giá trị trong bộ nhớ. Không giống biến, bạn phải gán các tên khác nhau, tất cả biến được lưu trong mảng có cùng tên. Bạn sẽ tham khảo những giá trị dữ liệu khác nhau bằng cách dùng một chỉ số.

Tất cả biến mà bạn đã làm việc trước đây là một tên biến duy nhất. Khi định nghĩa biến, bạn sẽ quyết định cả kiểu dữ liệu biến lẫn tên biến. Hình 11.1 chỉ ra những gì bạn đã định nghĩa một khi Visual Basic thi hành định nghĩa và phép gán biến như sau:

```
Dim Age As Integer
Dim Limit As Long
Dim Salary As Single
Dim DogName As String
' Store data in the variables
Age = 34
Limit = 40294
Salary = 982343.23 ' A Visual Basic programmer's pay
DogName = "Ralph"
```

Age	Salary
34	982343.23
Limits	**DogName**
40294	Ralph

Hình 11.1. *Các biến xuất hiện trong nhiều vùng nhớ khác nhau.*

Mặc dầu Visual Basic định nghĩa và gán các biến trở lại bộ nhớ, bạn vẫn không biết, thậm chí không quan tâm, nơi Visual Basic lưu các biến. Với những định nghĩa biến như thế, bạn cần bốn biến cho bốn kiểu dữ liệu và mỗi biến này có một tên khác nhau.

Mỗi biến thực hiện một mục đích khác nhau. Tuy nhiên, đôi khi bạn cần nhiều biến cùng kiểu dữ liệu để giữ các giá trị dữ liệu tương tự nhau. Ví dụ, một đại lý cần kiểm tra số lượng lớn tiền cấp giấy phép cho các giao dịch của ngày đó. Nếu có một trăm giao dịch, cách duy nhất để định nghĩa một trăm biến là định nghĩa và đặt tên khác nhau cho từng biến với những lệnh như sau :

```
Dim LicFee1, LicFee2, LicFee3, LicFee4 As Currency
Dim LicFee5, LicFee6, LicFee7, LicFee8 As Currency
Dim LicFee9, LicFee10, LicFee11, LicFee12 As Currency
' The rest of the variable definitions would follow
```

Khi bạn cần định nghĩa các biến vốn lưu giữ cùng loại các giá trị dữ liệu, chương trình của bạn sẽ trở nên nặng nề với những tên biến này. Việc điền vào các biến này là một vấn đề lớn. Bạn sẽ hỗ trợ 100 điều khiển *hộp nhập* (Text Box) trên một mẫu biểu chăng ?

Ngay cả khi bạn tìm ra cách điền 100 biến với các giá trị hàng ngày, thì bạn lấy tổng các giá trị đó như thế nào? Bạn sẽ phải viết thêm nhiều lệnh giống như sau:

```
DayTotal = LicFee1 + LicFee2 + LicFee3 + LicFee4
DayTotal = DayTotal + LicFee5 + LicFee6 + LicFee7
DayTotal = DayTotal + LicFee8 + LicFee9 + LicFee10
```

Thêm nữa, những biến bạn đã làm việc trước đây chỉ tốt cho các ứng dụng không cần xử lý nhiều tập hợp dữ liệu liên quan. Tuy nhiên, máy tính dùng để xử lý dữ liệu. Bạn có thể quay lại chương trước, ở đó sức mạnh của máy tính là khả năng lặp mã lệnh với tốc độ cao, việc xử lý một số lượng lớn dữ liệu rất nhanh chóng và không buồn chán.

Khái niệm mới

Mảng (array) là một danh sách các giá trị.

Visual Basic hỗ trợ việc sử dụng mảng để giúp bạn khỏi mệt mỏi nếu từng biến phải có một tên khác nhau. Thay vì định nghĩa các biến độc lập, bằng cách sử dụng nhiều lệnh Dim, bạn có thể định nghĩa một danh sách biến. Danh sách này được gọi là *mảng* (array), có các thuộc tính sau:

- Mỗi mục trong mảng được gọi là một thành phần.
- Mỗi thành phần trong mảng phải cùng kiểu dữ liệu.
- Danh sách có một tên và mỗi mục trong danh sách chứa dữ liệu của tên đó.
- Bạn sẽ thấy một chỉ số phụ để tham khảo các thành phần riêng biệt. Chỉ số phụ thường được gọi là chỉ số dưới của mảng (*subscript).*
- Thay vì sử dụng Dim, bạn sẽ sử dụng lệnh Static để định nghĩa mảng.

Ghi chú

Có hai cách khai báo mảng: dùng câu lệnh Dim và Global; đón đọc Chương 8.

Hình 11.1 là các biến không thể lưu trong mảng bởi vì các kiểu dữ liệu của chúng khác nhau. Tuy nhiên, tiền phí cấp phép cho 100 đại lý sẽ là ứng cử tuyệt vời cho việc tạo một mảng bởi vì tiền phí yêu cầu cùng loại dữ liệu (Currency).

Bạn sẽ sử dụng lệnh Static định nghĩa mảng. Static phần lớn giống Dim ngoại trừ Static định nghĩa mảng hơn là biến độc lập. Sau đây là dạng thức Static:

```
Static ArName(subMax) As DataType
```

ArName là tên mảng. *subMax* phải là một số mô tả tổng số thành phần mảng bạn cần. *DataType* là kiểu dữ liệu phù hợp với một trong các kiểu dữ liệu của Visual Basic, chẳng hạn như kiểu Single và Long.

Cảnh báo

Chẳng khác gì một loại biến bất kỳ, bạn phải định nghĩa mảng với Static *trước khi sử dụng chúng.*

Tóm tắt

Sau đây là cách đơn giản giúp đại lý định nghĩa 100 biến tiền phí cấp phép:

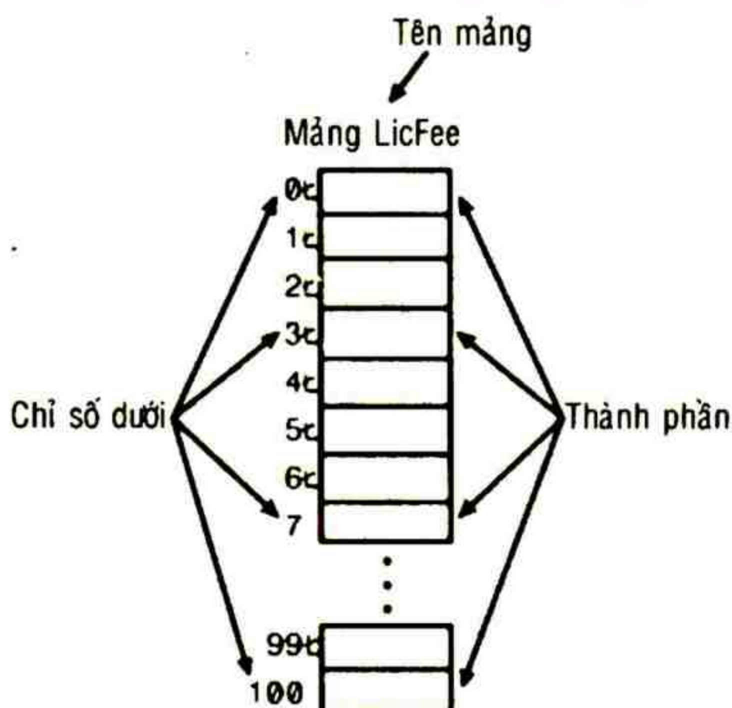
```
Dim LicFee(100) As Currency ' Defines 100 elements
```

Ôn lại

Bằng cách định nghĩa mảng dữ liệu, bạn có thể định nghĩa nhiều biến dữ liệu mà không phải gán cho từng biến một tên khác nhau và không phải định nghĩa từng biến riêng biệt. Mảng sẽ có một tên đơn lẻ và được lưu cùng với tất cả giá trị mảng tuần tự vào trong bộ nhớ. Bạn sẽ truy xuất từng giá trị bằng cách sử dụng một số chỉ mục duy nhất.

Kết quả

Hình 11.2 sẽ chỉ cách LicFee xuất hiện trong bộ nhớ.



Hình 11.2. *Các thành phần mảng được lưu trữ trong bộ nhớ với mỗi thành phần có một chỉ số duy nhất bắt đầu từ 0.*

Phân tích

Visual Basic tự động định nghĩa một thành phần mảng phụ với chỉ số là 0 khi bạn định nghĩa một mảng. Lệnh Option Base 1 mà bạn có thể đặt trong thủ tục (General) của bất kỳ cửa sổ Form nào sẽ báo cho Visual Basic biết để định nghĩa các mảng với chỉ số mảng bắt đầu từ 1. Không có Option Base 1, khi bạn yêu cầu 100 thành phần, Visual Basic sẽ đưa vào thêm chỉ số 0 trong mảng. Tuy nhiên, đa số lập trình viên Visual Basic quyết định bỏ qua chỉ số 0, và nếu bạn cũng làm như thế, bạn sẽ chiếm quá nhiều bộ nhớ bằng cách bỏ qua thành phần mảng phụ. Cuốn sách này sẽ luôn bỏ qua chỉ số 0 và làm việc với chỉ số bắt đầu của mảng là 1.

Visual Basic sẽ gán cho từng thành phần trong mảng một chỉ số để chương trình của bạn có thể phân biệt từng giá trị trong mảng với nhau. Tương tự như biến, Visual Basic sẽ gán số 0 với từng thành phần mảng, và chương trình của bạn sẽ ghi đè các giá trị 0 ban đầu này khi chương trình gán dữ liệu tới mảng.

Cảnh báo

Đừng bao giờ tham khảo một chỉ số mảng không tồn tại một mảng riêng biệt. Visual Basic sẽ phát ra một thông báo lỗi nếu chương trình của bạn thử làm một điều gì đó với LicFee(311) hay LicFee(–8) bởi vì 311 và –8 ở ngoài vùng chỉ số của LicFee, vì vùng này chỉ từ 0 tới 100.

ƯU ĐIỂM KHI SỬ DỤNG MẢNG

Khái niệm

Mảng sẽ loại trừ nhiều mã lệnh buồn tẻ. Việc sử dụng chỉ số mảng thích hơn là một tên biến khác, bạn có thể sử dụng vòng lặp For để nhảy qua từng chỉ số và truy xuất tuần tự các giá trị riêng của mảng.

Mảng làm đơn giản quá trình khởi tạo, in ấn, tính toán hay lưu dữ liệu. Ví dụ, nếu bạn định nghĩa một mảng 20 biến chuỗi lưu tên các khách hàng như sau:

```
Dim CustNames(20) As String
```

Bạn có thể khởi tạo từng giá trị của 20 chuỗi này với phép gán 20 hàm InputBox$() như sau:

```
CustName(1) = InputBox$("What is the next customer's name?")
CustName(2) = InputBox$("What is the next customer's name?")
CustName(3) = InputBox$("What is the next customer's name?")
CustName(4) = InputBox$("What is the next customer's name?")
CustName(5) = InputBox$("What is the next customer's name?")
CustName(6) = InputBox$("What is the next customer's name?")
CustName(7) = InputBox$("What is the next customer's name?")
CustName(8) = InputBox$("What is the next customer's name?")
CustName(9) = InputBox$("What is the next customer's name?")
' và tiếp tục như thế cho 20 tên
```

Bạn xem dùng vòng lặp For có dễ dàng hơn không ?

```
For Ctr = 1 To 20
    CustName(Ctr) = InputBox$("What is the next customer's name?")
Next Ctr
```

Biến Ctr nhảy qua từng giá trị từ 1 tới 20, gán kết quả của hàm InputBox$() cho từng thành phần mảng với 20 lần. Mã lệnh tương tự sau có thể hiển thị 20 tên trong các hộp thông báo (Message Box):

```
For Ctr = 1 To 20
 MsgBox CustName(Ctr)
Next Ctr
```

Có thể, hằng số hiển thị trong các *hộp nhận dữ liệu* (Input Box) và *hộp thông báo* (Message Box) đơn điệu và buồn tẻ, nhưng điều cốt

yếu bây giờ là việc lập trình mảng dễ dàng hơn là cung cấp biến có các tên khác nhau. Sử dụng vòng lặp For duyệt qua các giá trị trong mảng sẽ giảm nhẹ công việc với mảng đó.

Ghi chú

Dĩ nhiên, bạn không phải duyệt qua từng giá trị mảng khi chỉ cần làm việc với một vài thành phần trong mảng. Ví dụ, bạn có thể chỉ hiển thị khách hàng thứ 5 và thứ 19 với hai hộp thông báo sau:

```
MsgBox CustName(5)
MsgBox CustName(19)
```

Mảng sẽ cung cấp một cách tham khảo dữ liệu khác hơn là sử dụng tên biến khác nhau. Tuy nhiên, mỗi thành phần mảng được biết đến như là một biến duy nhất bởi tên mảng và chỉ số của nó. Vì vậy, bạn có thể làm việc với các thành phần mảng riêng biệt ngay khi chúng là các biến độc lập bằng cách truy xuất những chỉ số riêng của chúng.

Tóm tắt

Ví dụ 11.1 là một đoạn mã lệnh trong thủ tục tính giá trị trung bình tiền phí cấp phép của các giao dịch. Mảng có tên LicFee đã được điền các giá trị trước đó trong thủ tục. Những giá trị này có thể xuất phát từ mẫu biểu, từ người dùng với các *hộp nhận dữ liệu* (Input Box), hay từ một tập tin đĩa.

Ôn lại

Các chỉ số mảng cho phép bạn truy xuất một hay nhiều thành phần từ mảng. Điểm thuận lợi mà mảng cung cấp là cho phép chương trình của bạn duyệt qua toàn bộ mảng bằng cách sử dụng biến điều khiển của vòng lặp For dưới dạng chỉ số mảng.

Ví dụ 11.1. *Tính giá trị tiền phí trung bình từ một mảng.*

```
TotalFee = 0.0 ' TotalFee is defined as Currency
AvgFee = 0.0 ' AvgFee is defined as a Currency
For Ctr = 1 To 100 ' There are 100 license fees
TotalFee = TotalFee + LicFee(Ctr)
Next Ctr
' Now that the total is computed, calculate average
AvgFee = TotalFee / 100.0
```

Phân tích

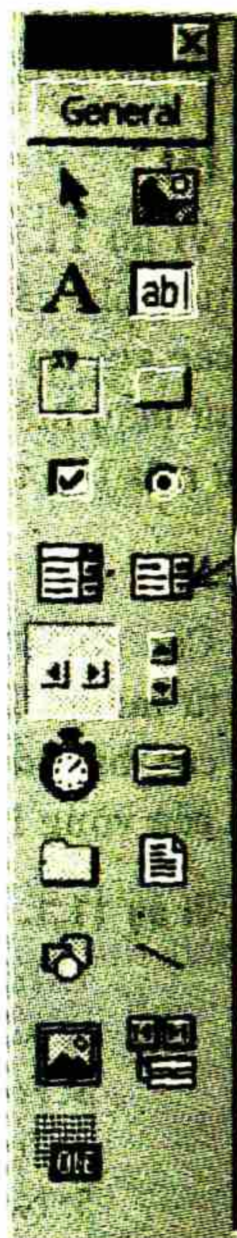
Trước khi đọc Ví dụ 11.1, hãy nghĩ cách tính giá trị trung bình các số. Đầu tiên bạn phải cộng các số với nhau, sau đó chia kết quả với tổng số hạng để có được giá trị trung bình. Quá trình đó là chính xác với những gì mã lệnh đã thực hiện. Vòng lặp For từ dòng 3 đến 5 sẽ chuyển các giá trị từ 1 đến 100 vào các biến có tên là Ctr trong mỗi lần lặp lại. Dòng 4 cộng từng giá trị của mảng, được điều khiển bởi vòng lặp For, tới biến TotalFee. Sau khi tất cả 100 số tiền phí cấp phép được cộng với nhau, dòng 7 lúc đó sẽ tính giá trị trung bình của 100 giá trị. Mã lệnh kế tiếp có thể hoặc hiển thị giá trị trung bình trong một nhãn (label) hoặc sử dụng giá trị đó cho một phép tính khác.

Hộp danh sách: ĐIỀU KHIỂN LÀM VIỆC GIỐNG MẢNG

Khái niệm

Điều khiển hộp danh sách (List Box) làm việc y như một mảng hoạt động trên mẫu biểu của bảng mà người dùng có thể cuộn qua. Hãy xem tất cả các mục trong danh sách. Thông thường, lập trình viên khởi tạo hộp danh sách (List Box) với dữ liệu từ mảng. Bạn không thể thêm các giá trị vào một điều khiển hộp danh sách qua thuộc tính cửa sổ Property, mà phải thêm các giá trị trong lúc chạy chương trình.

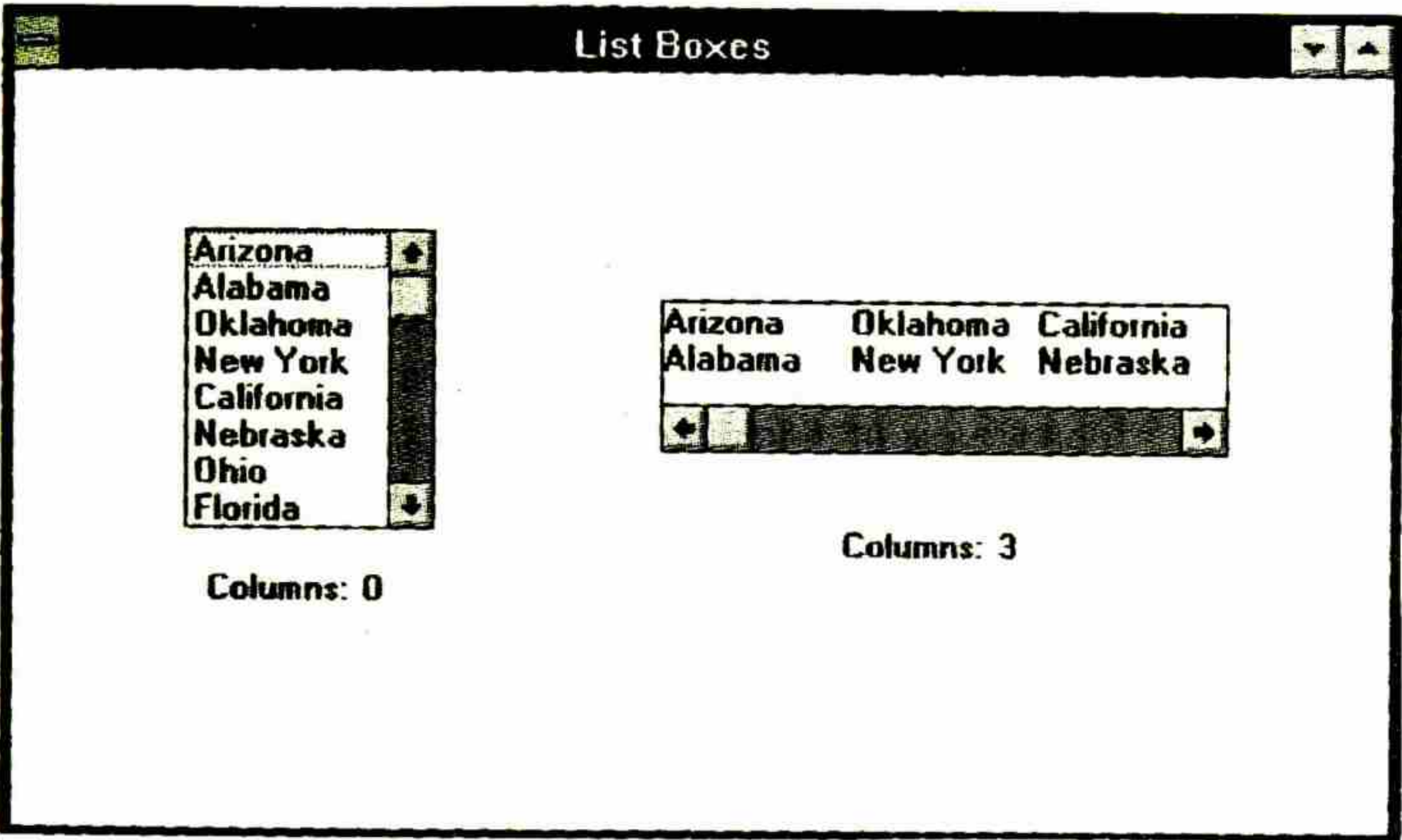
Hình 11.3 sẽ chỉ vị trí của điều khiển hộp danh sách (List Box) trên cửa sổ Toolbox. Điều khiển hộp danh sách sẽ hiển thị các giá trị mà người dùng có thể cuộn qua. Hộp danh sách không hiển thị các thanh cuốn nếu kích cỡ của hộp danh sách chỉ đủ lớn để hiển thị tất cả giá trị. Hãy nhớ rằng các hộp danh sách dùng để hiển thị dữ liệu, người dùng không thể thêm dữ liệu trực tiếp vào một hộp danh sách.



Hình 11.3. *Vị trí của điều khiển hộp danh sách.*

Lời nhắc

Hãy lưu ý cách điều khiển hộp danh sách làm việc. Có thể bạn muốn nạp và chạy project CONTROLS.VBP một lần nữa và cuộn qua các giá trị của điều khiển hộp danh sách trong ứng dụng.



Hình 11.4. *Hai hộp danh sách giống nhau nơi các xác lập thuộc tính Columns khác nhau.*

Bảng 11.1 bao gồm danh sách giá trị thuộc tính giúp bạn thiết đặt điều khiển hộp danh sách. Các điều khiển có nhiều thuộc tính như nhau.

Bảng 11.1. *Các thuộc tính hộp danh sách*

Thuộc tính	Diễn giải
BackColor	Màu nền của hộp danh sách. Đó là một số thập lục phân trình bày một trong hàng ngàn giá trị màu Windows có thể sở hữu. Bạn chọn từ một bảng màu được Visual Basic hiển thị khi thiết đặt thuộc tính BackColor. Màu nền mặc định giống với màu nền mặc định của mẫu biểu.

Columns	Nếu là 0 (giá trị mặc định), hộp danh sách sẽ cuộn dọc trong một cột đơn. Nếu là 1 hay hơn, các mục trong hộp danh sách sẽ xuất hiện với số cột đã xác định (một hay nhiều cột) mà người dùng cuộn hộp danh sách theo chiều ngang để xem tất cả các mục nếu cần. Hình 11.4 minh họa hai hộp danh sách giống hệt nhau, một với thuộc tính Columns bằng 0 và một với thuộc tính Columns bằng 3.
DragIcon	Biểu tượng sẽ xuất hiện khi người dùng kéo điều khiển hộp danh sách quanh mẫu biểu. (Hiếm khi bạn cho phép người dùng di chuyển một điều khiển hộp danh sách, cho nên các xác lập thuộc tính Drag... không thích hợp.)
DragMode	Hoặc bằng 1 cho yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo các điều khiển) hoặc 0 (giá trị mặc định) cho việc kéo mouse tự động, nghĩa là người dùng không thể kéo hộp danh sách, nhưng bạn bằng mã lệnh có thể khởi tạo việc kéo nếu cần.
Enabled	Nếu là True (mặc định), hộp danh sách có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng việc xử lý biến cố cho điều khiển riêng biệt.
FontBold	True (mặc định) tương ứng với các giá trị sẽ hiển thị ở dạng chữ đậm; ngược lại là False.
FontItalic	True (mặc định) tương ứng với các giá trị sẽ hiển thị ở dạng chữ nghiêng; ngược lại là False.
FontName	Tên kiểu văn bản của hộp danh sách. Thông thường, bạn sẽ sử dụng tên của một phông chữ TrueType trong Windows.
FontSize	Kích cỡ tính theo point của phông chữ được sử dụng cho các giá trị hộp danh sách.
FontStrikethru	True (mặc định) tương ứng với các giá trị sẽ hiển thị ở dạng ký tự bị gạch ngang (các ký tự có một đường kẻ ngang qua chúng); ngược lại là False.
FontUnderline	True (mặc định) tương ứng với các giá trị sẽ hiển thị ở dạng gạch dưới; ngược lại là False.
ForeColor	Màu của các giá trị trong hộp danh sách.
Height	Chiều cao tính bằng twip, của điều khiển hộp danh sách.

HelpContextID	Nếu bạn thêm trợ giúp cảm ngữ cảnh cao cấp vào ứng dụng của bạn, giá trị HelpContextID cung cấp một số xác định văn bản trợ giúp.
Index	Nếu hộp danh sách là một phần của một điều khiển mảng, thuộc tính Index sẽ cung cấp chỉ số cho từng điều khiển hộp danh sách riêng biệt (xem bài tiếp theo).
Left	Khoảng cách tính bằng twip từ góc trái cửa sổ Form tới góc trái điều khiển hộp danh sách (List Box)
MousePointer	Hình dạng con trỏ mouse sẽ thay đổi nếu người dùng di chuyển con trỏ mouse trên điều khiển hộp danh sách. Các giá trị có thể từ 0 đến 12 và trình bày những hình dạng khác nhau mà con trỏ mouse có thể lấy. (Xem Chương 12.)
MultiSelect	Nếu là 0–None (mặc định), người dùng chỉ có thể chọn một mục trong hộp danh sách. Nếu là 1–Simple, người dùng có thể chọn nhiều mục bằng cách nhấp mouse hay nhấn phím Spacebar trên các mục trong danh sách. Nếu là 2–Extended, người dùng có thể chọn nhiều mục bằng cách sử dụng Shift+nhấp mouse và Shift+mũi tên để mở rộng việc chọn từ mục trước tới mục hiện hành, Ctrl+nhấp mouse để chọn hay bỏ chọn một mục từ danh sách.
Name	Tên của điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên List1, List2, và tiếp tục như thế khi bạn thêm tuần tự các điều khiển hộp danh sách vào mẫu biểu.
Sorted	Nếu là True, Visual Basic sẽ không hiển thị các giá trị hộp danh sách được sắp xếp theo giá trị số hay thứ tự abc. Nếu là False (mặc định), các giá trị sẽ xuất hiện với thứ tự mà chương trình bổ sung chúng vào danh sách.
TabIndex	Thứ tự tab tiêu điểm bắt đầu từ 0 và tăng mỗi lần bạn thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm (focus) bằng cách thay đổi thuộc tính TabIndex của điều khiển thành các giá trị khác. Không có hai điều khiển nào trên cùng một mẫu biểu lại có cùng giá trị TabIndex.
TabStop	Nếu là True, người dùng có thể nhấn Tab để chuyển tiêu điểm tới hộp danh sách này. Nếu là False, hộp danh sách không thể nhận tiêu điểm.

Tag	Không được Visual Basic sử dụng. Đây là tiện ích dành cho lập trình viên xác định chú thích áp dụng vào điều khiển hộp danh sách.
Top	Khoảng cách twip từ cạnh trên cùng của điều khiển hộp danh sách tới cạnh trên cùng của mẫu biểu.
Visible	True hay False, chỉ định người dùng có thể thấy điều khiển hộp danh sách hay không (và như vậy mới có thể sử dụng được).
Width	Chiều rộng tính bằng twip của hộp danh sách.

Khi đặt một điều khiển hộp danh sách (List Box) trên mẫu biểu, hãy quyết định cách bạn muốn hộp danh sách thực hiện để định kích cỡ điều khiển cho phù hợp với kích cỡ mẫu biểu. Hãy nhớ rằng nếu tất cả giá trị hộp danh sách không chứa đủ trong hộp danh sách, Visual Basic sẽ thêm các thanh cuốn vào hộp danh sách sao cho người dùng có thể cuộn qua các giá trị.

Bảng 11.2 chứa tất cả biến cố list box mà bạn có thể lập trình và sử dụng trong một chương trình. Tuy nhiên, bạn ít khi viết thủ tục biến cố cho điều khiển hộp danh sách. Phần lớn thời gian, bạn cho phép người dùng cuộn qua các giá trị hộp danh sách để xem thông tin họ cần; chương trình không phải đáp ứng các biến cố list box thường xuyên như khi đáp ứng *nút lệnh* (Command Button) và *hộp nhập* (Text Box).

Bảng 11.2. *Các biến cô của điều khiển hộp danh sách.*

Biến cố	Diễn giải
Click	Xảy ra khi người dùng nhấp điều khiển hộp danh sách.
DblClick	Xảy ra khi người dùng nhấp đúp điều khiển hộp danh sách.
DragDrop	Xảy ra khi một thao tác kéo hộp danh sách hoàn thành.
DragOver	Xảy ra trong suốt thời gian kéo.
GotFocus	Xảy ra khi hộp danh sách nhận tiêu điểm.
KeyDown	Xảy ra khi người dùng nhấn một phím đồng thời thuộc tính KeyPreview được thiết đặt là True cho các điều khiển trên mẫu biểu; ngược lại mẫu biểu sẽ nhận biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn một phím trên hộp danh sách.

KeyUp	Xảy ra khi người dùng thả một phím trên hộp danh sách.
LostFocus	Xảy ra khi hộp danh sách mất tiêu điểm cho đối tượng khác.
MouseDown	Xảy ra khi người dùng nhấn một nút mouse trên hộp danh sách.
MouseMove	Xảy ra khi người dùng di chuyển mouse trên hộp danh sách.
MouseUp	Xảy ra khi người dùng thả một nút mouse trên hộp danh sách.

Bảng 11.3 là danh sách các phương thức điều khiển hộp danh sách dùng để khởi tạo, phân tích và gỡ bỏ các khoản mục từ một điều khiển hộp danh sách. Phương thức hoạt động giống những chương trình nhỏ thi hành trên điều khiển. Sau đây là dạng sử dụng của phương thức trên hộp danh sách có tên là stItems: lstIems.AddItem “Arizona”.

Tên điều khiển luôn đứng trước phương thức và toán tử chấm. Dữ liệu bất kỳ được phương thức cần đến sẽ xuất hiện bên phải phương thức.

Bảng 11.3. *Các phương thức điều khiển hộp danh sách*

Tên phương thức	Diễn giải
AddItem	Thêm một mục vào hộp danh sách.
Clear	Xóa tất cả các mục trong hộp danh sách.
List	Một mảng chuỗi giữ từng mục trong hộp danh sách.
ListCount	Tổng số mục trong hộp danh sách.
RemoveItem	Gỡ bỏ một mục khỏi hộp danh sách.
Selected	Xác định xem người dùng đã chọn một mục riêng biệt trong hộp danh sách chưa.

Sử dụng phương thức AddItem để bổ sung các thuộc tính vào điều khiển hộp danh sách. Giả sử bạn muốn thêm một vài tên tiểu bang vào hộp danh sách có tên là lstStates. Mã lệnh sau sẽ thực hiện việc đó.

```
' Add several states to a list box control
lstStates.AddItem "Arizona"
lstStates.AddItem "Alabama"
```

```
lstStates.AddItem "Oklahoma"
lstStates.AddItem "New York"
lstStates.AddItem "California"
lstStates.AddItem "Nebraska"
lstStates.AddItem "Ohio"
lstStates.AddItem "Florida"
lstStates.AddItem "Texas"
lstStates.AddItem "South Dakota"
lstStates.AddItem "Nevada"
lstStates.AddItem "Illinois"
lstStates.AddItem "New Mexico"
```

Mã lệnh này phần lớn xuất hiện trong thủ tục biến cố Form_Load() để các hộp danh sách được khởi tạo với những giá trị của chúng trước khi mẫu biểu xuất hiện và trước khi nhìn thấy các hộp danh sách trên mẫu biểu.

Ghi chú

Hộp danh sách hoạt động khá giống một mảng.

Từng mục trong hộp danh sách có một chỉ số như của từng phần tử trong một mảng có một chỉ số. Mục đầu tiên bạn thêm vào hộp danh sách có chỉ số là 0, mục thứ 2 có chỉ số là 1, và v.v. Vì vậy, để gỡ bỏ mục thứ ba khỏi hộp danh sách, mã lệnh của bạn có thể áp dụng phương thức Removeltem như sau :

```
lstStates.RemoveItem(2) ' 3rd item has a subscript of 2
```

Chú ý

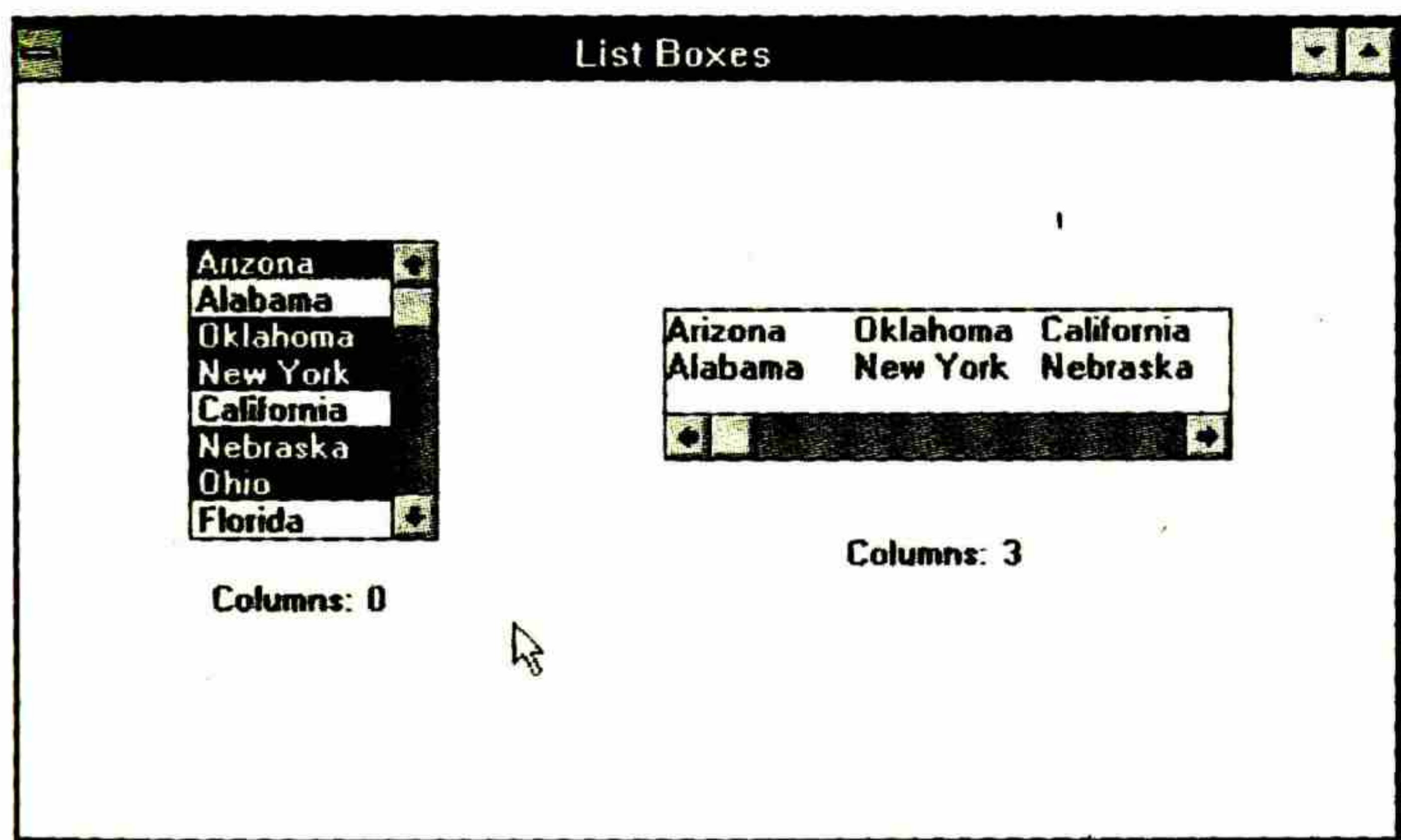
Phải nhớ rằng, khi bạn gỡ bỏ các mục khỏi một hộp danh sách, các chỉ số của những mục còn lại sẽ điều chỉnh theo. Vì vậy, nếu một hộp danh sách có 7 mục, mỗi mục có một chỉ số từ 0 đến 6. Nếu bạn gỡ bỏ mục thứ 4, các mục hộp danh sách sau đó sẽ đánh số từ 0 tới 5; chỉ số 5 sẽ dành cho mục có chỉ số 6 trước khi phương thức Removeltem gỡ bỏ mục 4.

Nếu bạn muốn gỡ bỏ tất cả các mục khỏi hộp danh sách, hãy sử dụng phương thức Clear. Phương thức đơn giản sau sẽ gỡ bỏ tất cả tên tiểu bang khỏi hộp danh sách:

```
lstStates.Clear ' Remove all items
```

Bạn có thể gán các mục riêng từ một điều khiển hộp danh sách chứa dữ liệu bằng cách dùng phương thức List. Bạn phải cất các giá trị hộp danh sách trong biến chuỗi trừ khi bạn chuyển các mục hộp danh sách thành kiểu dữ liệu số bằng cách sử dụng hàm Val(). Những lệnh gán sau sẽ lưu mục đầu tiên và mục thứ tư trong hộp danh sách vào hai biến chuỗi:

```
FirstStringVar = lstStates.List(0)
SecondStringVar = lstStates.List(3)
```



Hình 11.5. *Một hộp danh sách với thuộc tính MultiSelect ấn định bằng 1 hoặc 2.*

Phương thức ListCount luôn cho biết tổng số mục trong điều khiển hộp danh sách. Ví dụ, câu lệnh sau sẽ lưu số mục hộp danh sách vào một biến số có tên là Num:

```
Num = lstStates.ListCount
```

Phương thức Selected trả lại hoặc giá trị true hoặc false tùy thuộc người dùng đã chọn một mục trong hộp danh sách. Phương thức Selected trả lại true có khả năng có nhiều mục trong hộp danh sách được chọn nếu thuộc tính MultiSelect được thiết đặt hoặc bằng 1–Simple hoặc bằng 2–Extended. Các thuộc tính này cho thấy người dùng có thể chọn nhiều mục một lần. Hình 11.5 minh họa một hộp danh sách với nhiều mục được chọn cùng lúc.

Tóm tắt

Mã lệnh trong Ví dụ 11.2 sẽ lưu từng mục được chọn trong hộp danh sách có tên lstStates vào mảng chuỗi. Mảng chuỗi được định nghĩa với đủ thành phần để giữ tất cả các mục nếu người dùng chọn 50 mục đầu tiên đã được thêm vào hộp danh sách.

Ôn lại

Một điều khiển hộp danh sách giữ một hay nhiều giá trị mà người dùng có thể cuộn qua. Không giống nhiều điều khiển khác, người dùng chỉ có thể nhìn và chọn các mục trong hộp danh sách, nhưng không thêm các mục vào hộp danh sách được. Tuy nhiên, qua các phương thức, bạn có thể thêm mục, xóa mục, và kiểm tra mục đã chọn trong hộp danh sách bằng việc sử dụng phương thức trong thủ tục mã lệnh của bạn.

Ví dụ 11.2. *Lưu tất cả giá trị được chọn vào một mảng chuỗi.*

```
1: Dim SelectStates(50) As String
2: Dim StSub As Integer ' State array subscript
3: StSub = 1
4: For Ctr = 0 To 49 ' 50 states in all
5: If (lstStates.Selected(Ctr) = True) Then
6: SelectStates(StSub) = lstStates.List(Ctr)
7: StSub = StSub + 1
8: End If
10: Next Ctr
```

Phân tích

Hai dòng 1 và 2 phải định nghĩa chỉ số đóng vai trò tìm mảng chuỗi các tiểu bang. Dòng 3 sẽ khởi tạo chỉ số bắt đầu là 1 bởi vì chỉ số được bỏ qua trong các chương trình thuộc sách này; đây là một tiêu chuẩn được đa số lập trình viên Visual Basic y theo.

Dòng 4 đến 10 có chứa một vòng lặp duyệt qua tất cả 50 tiểu bang được lưu trong hộp danh sách trước đó (có lẽ trong thủ tục biến cố Form_Load()). Nếu người dùng chọn một tiểu bang (dòng 5 sẽ kiểm tra việc chọn lựa), phương thức Selected sẽ trả lại kết quả true và dòng 6

sẽ lưu mục được chọn trong hộp danh sách vào mảng chuỗi. Dòng 7 tăng chỉ số của mảng chuỗi cho phép gán có thể xảy ra tiếp theo trong vòng lặp.

Hộp Combo

Khái niệm



Hộp combo (Combo Box) làm việc khá giống hộp danh sách, có khác chăng chỉ là người dùng có thể thêm nội dung vào hộp combo lúc chạy chương trình. Có ba loại hộp combo được xác định bởi thuộc tính Style. Các phương thức AddItem và RemoveItem khá phổ biến cho các hộp combo, mặc dầu tất cả điều khiển hộp danh sách trong bài trước đều áp dụng được với hộp combo.

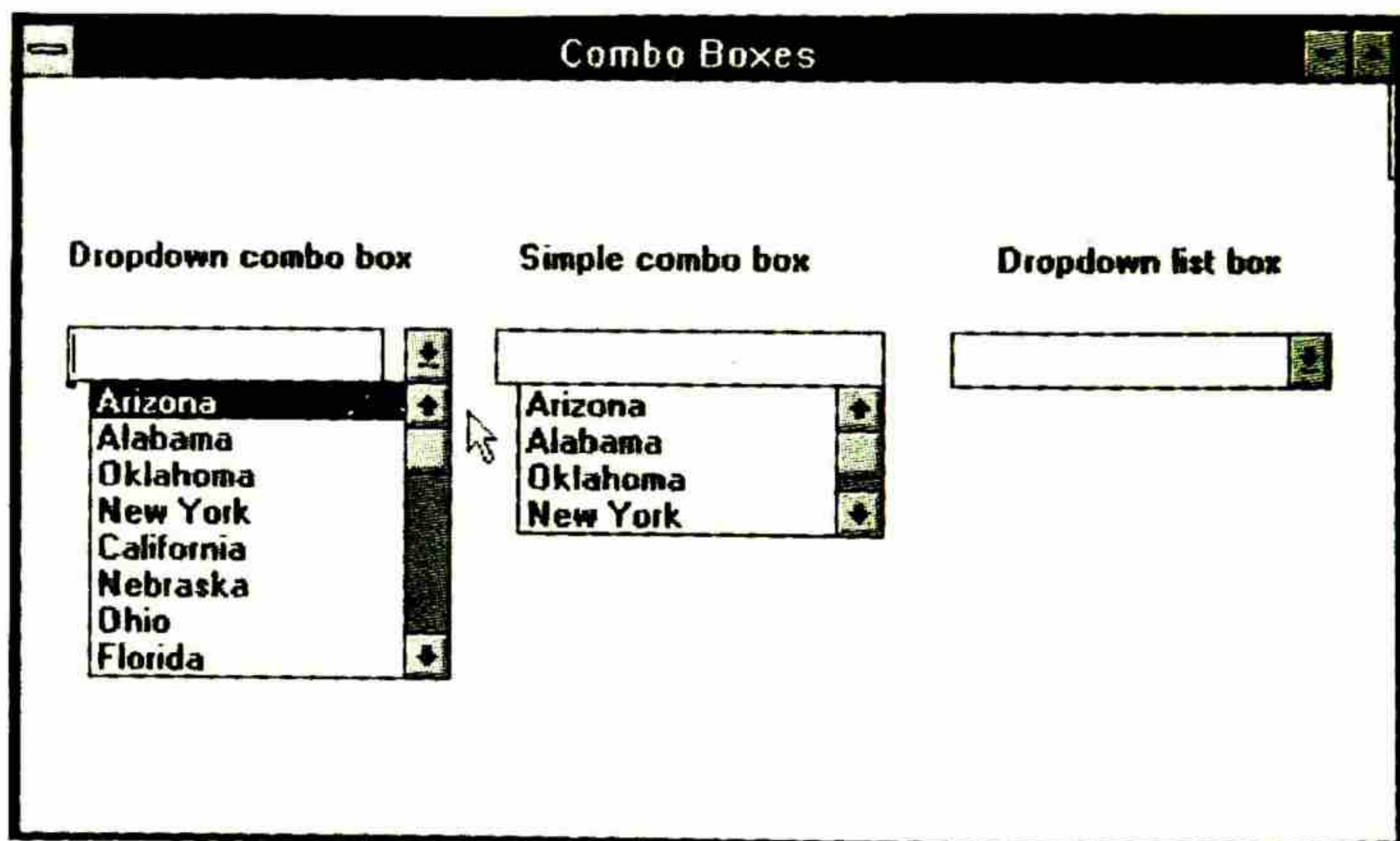
Hình 11.6 cho thấy vị trí của điều khiển hộp combo trên cửa sổ Toolbox. Không có gì phải lo ngại với các loại hộp combo bạn muốn đặt trên mẫu biểu. Bạn sẽ sử dụng cùng điều khiển hộp combo trên hộp công cụ để thêm hộp combo vào mẫu biểu.

Hình 11.6. *Vị trí của điều khiển hộp combo.*

Có 3 loại hộp combo:

- *hộp combo cuộn xuống* (Dropdown Combo Box) chỉ chiếm một dòng đơn trên mẫu biểu trừ khi người dùng mở hộp combo (bằng cách nhấn mũi tên xuống của hộp combo) để xem giá trị bổ sung. Người dùng có thể nhập vào những mục bổ sung ở đâu đó trên hộp combo xổ xuống và chọn các mục từ hộp combo.

- *hộp combo đơn giản* (Simple Combo Box) luôn luôn hiển thị các mục như chúng đang ở trong một hộp danh sách (List Box). Người dùng dễ dàng bổ sung các mục vào danh sách hộp combo.
- *hộp danh sách cuộn xuống* (Dropdown List Box) là một dạng hộp danh sách đặc biệt, người dùng không thể nhập vào các mục mới, nhưng các mục xuất hiện bình thường đó được gắn vào một dòng đơn cho đến khi người dùng nhấp nút mũi tên xuống để mở hộp danh sách với kích thước đầy đủ của nó. Về mặt kỹ thuật, hộp danh sách cuộn xuống là điều khiển hộp combo nhưng làm việc giống với hộp danh sách hơn. Lý do hộp danh sách cuộn xuống nằm trong nhóm các điều khiển hộp combo quen thuộc là khi bạn đặt hộp danh sách cuộn xuống trên các mẫu biểu bằng cách nhấp điều khiển hộp combo và ấn định thuộc tính hộp combo thích hợp (Style).



Hình 11.7 *Ba kiểu hộp combo.*

Hình 11.7 minh họa ba loại hộp combo. Mỗi loại có chứa tên các tiểu bang mà bạn đã thấy trước đó trong hộp danh sách. Hộp combo đầu tiên, hộp combo cuộn xuống, thường đóng; khi người dùng nhấp mũi tên xuống của hộp combo, hộp combo sẽ mở. Hộp combo thứ ba, hộp

danh sách cuộn xuống, không được mở phía bên trái. Nếu người dùng mở hộp danh sách cuộn xuống, người dùng sẽ thấy danh sách tên các tiểu bang nhưng không thể thêm các tên tiểu bang mới vào bởi vì không có vùng nhập dữ liệu nào trong hộp danh sách cuộn xuống.

Bảng 11.4 chứa nội dung mô tả từng giá trị thuộc tính. Danh sách hộp combo có thể thiết đặt trong một cửa sổ Property.

Bảng 11.4. *Các thuộc tính hộp combo*

Thuộc tính	Diễn giải
BackColor	Màu nền của hộp combo. Đây là một số thập lục phân trình bày một trong số hàng ngàn giá trị màu có thể có trong Windows. Bạn dễ dàng chọn từ một bảng màu được hiển thị bởi Visual Basic khi thiết đặt thuộc tính BackColor. Màu nền mặc định giống màu nền mặc định của mẫu biểu.
DragIcon	Biểu tượng sẽ xuất hiện khi người dùng kéo điều khiển hộp combo trên mẫu biểu. (Năm thì mười họa bạn mới cho phép người dùng di chuyển một điều khiển hộp combo, cho nên thiết đặt thuộc tính Drag... thường không thích hợp.)
DragMode	Hoặc bằng 1 khi phải kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển) hoặc 0 (mặc định) cho việc kéo mouse tự động, nghĩa là người dùng không thể kéo điều khiển hộp combo nhưng bạn, thông qua mã lệnh, có thể khởi tạo việc kéo nếu cần.
Enabled	Nếu là True (mặc định), điều khiển hộp combo có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng việc xử lý biến cố cho điều khiển riêng biệt đó.
FontBold	True (mặc định) để hiển thị các giá trị combo ở dạng chữ đậm; ngược lại là False.
FontItalic	True (mặc định) để hiển thị các giá trị combo ở dạng chữ nghiêng; ngược lại là False.
FontName	Tên kiểu văn bản của hộp combo. Thông thường, bạn sẽ sử dụng tên của một phông chữ TrueType trong Windows.
FontSize	Kích cỡ tính theo point của phông chữ được sử dụng cho các giá trị hộp combo.

FontStrikethru	True (mặc định) hiển thị các giá trị combo ở dạng chữ bị gạch bỏ (nghĩa là mỗi ký tự có một đường gạch ngang qua nó); ngược lại là False.
FontUnderline	True (mặc định) để hiển thị các giá trị hộp combo ở dạng gạch dưới các ký tự ; ngược lại là False.
ForeColor	Màu của các giá trị bên trong hộp combo.
Height	Chiều cao tính bằng twip, của điều khiển hộp combo.
HelpContextID	Nếu bạn thêm trợ giúp cảm ngữ cảnh cao cấp vào ứng dụng của bạn), giá trị HelpContextID sẽ cung cấp một giá trị xác định cho văn bản trợ giúp.
Index	Nếu điều khiển hộp combo là một phần của mảng điều khiển, thuộc tính Index sẽ cung cấp số chỉ số cho từng điều khiển hộp combo riêng biệt.
Left	Khoảng cách tính bằng twip từ góc trái cửa sổ Form tới góc trái điều khiển hộp combo.
MousePointer	Hình dạng con trỏ mouse sẽ thay đổi nếu người dùng di chuyển con trỏ mouse trên điều khiển hộp combo (Combo Box). Các giá trị có thể từ 0 đến 12 và trình bày các hình dạng khác nhau của con trỏ mouse.
Name	Tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên Combo1, Combo2, và tiếp tục như thế khi bạn thêm các điều khiển hộp combo lần lượt vào mẫu biểu.
Sorted	Nếu là True, Visual Basic sẽ không hiển thị các giá trị hộp combo được sắp xếp theo giá trị số hay thứ tự abc. Nếu là False (mặc định), các giá trị sẽ xuất hiện theo cùng thứ tự mà chương trình đã thêm chúng vào danh sách.
Style	Mặc định, 0–Dropdown Combo, đưa ra một điều khiển hộp combo cuộn xuống. 1–Simple Combo chuyển hộp combo thành một điều khiển hộp combo đơn giản. 2–Dropdown list chuyển hộp combo thành một điều khiển hộp danh sách cuộn xuống.
TabIndex	Thứ tự tab tiêu điểm bắt đầu bằng 0 và tăng mỗi lần bạn thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm (focus) bằng cách thay đổi giá trị TabIndex của điều khiển thành các giá trị khác. Không có hai điều khiển trên cùng mẫu biểu có thể có cùng giá trị TabIndex.

TabStop	Nếu là True, người dùng có thể nhấn Tab để di chuyển tiêu điểm tới hộp combo này. Nếu là False, hộp combo không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Đây là tiện ích dành cho lập trình viên xác định chú thích áp dụng cho điều khiển hộp combo.
Text	Giá trị ban đầu duy nhất người dùng nhìn thấy trong hộp combo.
Top	Khoảng cách tính bằng twip từ cạnh trên cùng điều khiển hộp combo tới cạnh trên cùng của mẫu biểu.
Visible	True (mặc định) hoặc False, chỉ ra liệu người dùng có thể thấy (và vì vậy sẽ sử dụng) điều khiển hộp combo.
Width	Độ rộng tính bằng twip mà điều khiển hộp combo chiếm.

Ghi chú

Lưu ý rằng không có thuộc tính MultiSelect trong hộp combo (Combo Box) như trong các điều khiển hộp danh sách (List Box). Người dùng có thể chọn một mục duy nhất trong hộp combo tại một thời điểm.

Bảng 11.5 giới thiệu một danh sách biến cố hộp combo. Từ đó bạn có thể viết các thủ tục biến cố phù hợp khi chương trình của bạn phải phản ứng trở lại tác động của người dùng lên một hộp combo.

Bảng 11.5. *Các biến cố của điều khiển hộp combo*

Biến cố	Diễn giải
Change	Xảy ra khi người dùng thay đổi giá trị vùng dữ liệu nhập vào của hộp combo cuộn xuống hay hộp combo đơn giản; không có tác dụng với hộp danh sách cuộn xuống bởi vì người dùng không thể thay đổi dữ liệu trong chúng.
Click	Xảy ra khi người dùng nhấp điều khiển hộp combo.
DblClick	Xảy ra khi người dùng nhấp đúp điều khiển hộp combo.
DragDrop	Xảy ra khi một thao tác kéo của hộp combo hoàn thành.
DragOver	Xảy ra trong suốt thao tác kéo.
DropDown	Xảy ra khi người dùng mở một hộp combo cuộn xuống hay một hộp danh sách cuộn xuống.

GotFocus	Xảy ra khi hộp combo nhận tiêu điểm.
KeyDown	Xảy ra khi người dùng nhấn một phím đồng thời thuộc tính KeyPreview được ấn định là True cho các điều khiển trên mẫu biểu; ngược lại, mẫu biểu sẽ nhận biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn một phím trên hộp combo.
KeyUp	Xảy ra khi người dùng thả một phím trên hộp combo.
LostFocus	Xảy ra khi hộp combo chuyển tiêu điểm sang một điều khiển khác.

Điều khiển hộp combo (Combo Box) hỗ trợ cùng phương thức mà điều khiển hộp danh sách hỗ trợ. Vì vậy, có thể thêm vào, gỡ bỏ, đếm, và chọn các mục từ hộp combo nếu bạn áp dụng các phương thức trong Bảng 11.3.

Tóm tắt

Ví dụ 11.3 chứa các thủ tục biến cố của nút lệnh. Khác với điều khiển hộp nhập (Text Box), bạn cần cung cấp cho người dùng một số cơ chế nhập liệu, chẳng hạn như nút lệnh, báo cho chương trình biết khi sử dụng phương thức AddItem để thêm một mục vào hộp combo.

Ghi chú

Ví dụ 11.3 giả sử thủ tục khác, như thủ tục Form_Load(), đã thêm các giá trị ban đầu vào hộp combo.

Ôn lại

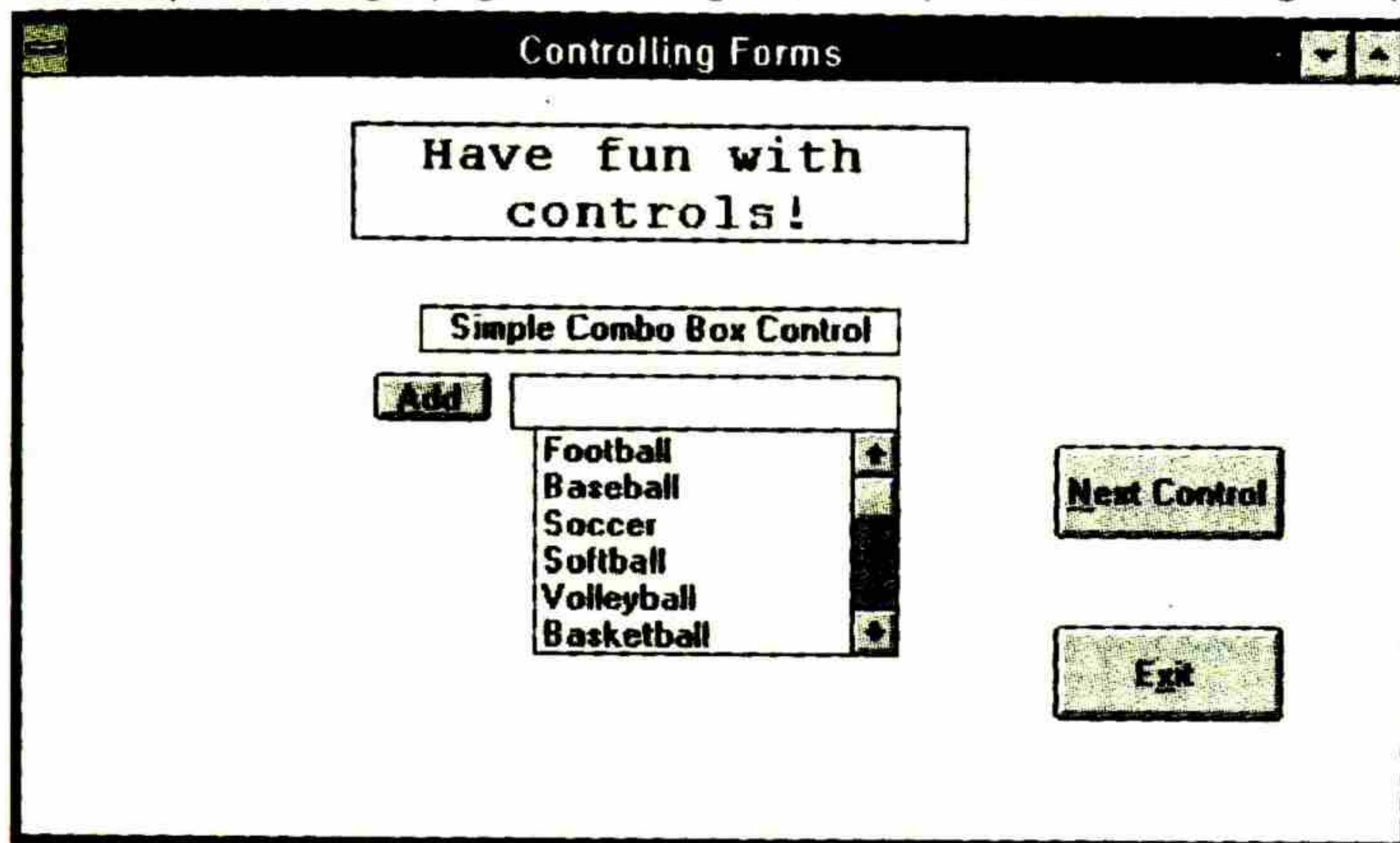
Ba loại hộp combo (Combo Box) đưa ra ba điều khiển tương tự nhau mà chương trình của bạn có thể tác động bằng cách giới thiệu những danh sách giá trị và tùy chọn, cho phép người dùng nhập giá trị vào danh sách. Hai hộp combo là các danh sách cuộn xuống không chiếm khoảng trống cho đến khi người dùng mở hộp combo.

Ví dụ 11.3. *Thủ tục biến cố của một nút lệnh bổ sung khoản mục vào hộp combo.*

```
Sub cmdSimple_Click ()
' Add the user's item to the simple combo
comSCtrl.AddItem comSCtrl.Text
comSCtrl.Text = ""
comSCtrl.SetFocus
End Sub
```

Kết quả

Hình 11.8 là một màn hình bạn đã gặp ở Chương 2. Ứng dụng CONTROLS.VBP minh họa một hộp combo đơn giản và chỉ cách bạn có thể khởi tạo một ứng dụng để bổ sung khoản mục vào danh sách giá trị.



Hình 11.8. *Thêm dữ liệu vào một điều khiển hộp combo đơn giản.*

Phân tích

Nút lệnh trong Hình 11.8 có tên là cmdSimple, nên việc nhấp nút lệnh sẽ thi hành thủ tục biến cố trong Ví dụ 11.3. Dòng 3 lưu giá trị thuộc tính Text của hộp combo với danh sách các mục của hộp combo đó. Hộp combo sẽ không chứa một vùng nhập liệu nào của người dùng ở phần trên hộp combo cho đến khi một phương thức AddItem sẽ thêm nội dung vào danh sách đó. Thuộc tính Text luôn luôn giữ giá trị hiện hành, nhưng phương thức AddItem phải thêm giá trị đó vào danh sách.

Ngay khi thêm dữ liệu nhập vào của người dùng, dòng 4 sẽ xóa phần dữ liệu trong vùng nhập của hộp combo. Sau cùng, văn bản của người dùng bây giờ sẽ xuất hiện ở phần dưới danh sách của hộp combo (cảm ơn dòng 3), cho nên dòng 4 sẽ xóa vùng nhập liệu vào chờ nội dung nhập tiếp theo. Thêm vào đó, dòng 5 đặt tiêu điểm (focus) trở lại hộp combo (tiêu điểm sẽ xuất hiện trong vùng nhập liệu vào mà dòng 4 đã xóa) để người dùng sẵn sàng thêm một mục bổ sung vào hộp combo.

Bài tập

Kiến thức tổng quát

1. Mảng có thể giúp bạn xử lý một số lượng lớn dữ liệu như thế nào?
2. Thành phần mảng là gì?
3. Thế nào là chỉ số mảng?
4. Nếu một mảng chứa 20 thành phần, có bao nhiêu tên cho mảng đó?
5. Nếu một mảng chứa 20 thành phần, có bao nhiêu chỉ số cho mảng (giả sử có lệnh Option Base trong cửa sổ Code).
6. Đúng hay Sai: Mỗi thành phần trong mảng phải cùng kiểu dữ liệu.
7. Bạn có thể phân biệt các thành phần trong mảng như thế nào?
8. Lệnh nào sẽ định nghĩa mảng bên trong các thủ tục biến cố?
9. Giả sử có một lệnh Option Base 1 trong cửa sổ Code, chỉ số bắt đầu cho tất cả mảng mà bạn định nghĩa lần lượt trong chương trình là gì?
10. Vòng lặp nào làm việc tốt nhất khi xử lý mảng?
11. Những điều khiển nào làm việc giống mảng?
12. Đúng hay Sai: Người dùng có thể nhập những giá trị mới trong hộp danh sách.
13. Đúng hay Sai: Người dùng có thể chọn nhiều giá trị trong một hộp danh sách.
14. Đúng hay Sai: Một hộp danh sách có thể có nhiều cột.
15. Tại sao phương thức lại quan trọng với các hộp danh sách và hộp combo?
16. Thủ tục biến cố nào lập trình viên thường dùng để khởi tạo các hộp danh sách và hộp combo?
17. Đúng hay Sai: Có ba loại hộp combo.

18. Đúng hay Sai: Có ba loại điều khiển hộp combo trên hộp công cụ.
19. Đúng hay Sai: Các điều khiển hộp danh sách cũng làm việc tốt cho các hộp combo.
20. Đúng hay Sai: Các điều khiển hộp danh sách và hộp combo có chỉ số như mảng vậy.
21. Phương thức nào cho biết một mục hộp danh sách có được chọn?
22. Đúng hay Sai: Một hộp combo đơn giản có thể mở bằng cách nhấp mouse vào mũi tên xuống của hộp combo đơn giản.

Viết mã lệnh...

23. An Huy muốn bảo đảm tất cả các mục trong hộp danh sách của anh ấy được sắp xếp. Thuộc tính nào An Huy nên thiết đặt là 1–Sorted?
24. Giả sử bạn muốn định nghĩa một mảng An Huy lưu tuổi của 3 học sinh trung học trong một danh sách. Hãy viết lệnh Static để định nghĩa mảng đó.
25. Thuộc tính nào của hộp danh sách bạn sẽ định là 1 hoặc 2 nếu bạn muốn cho phép người dùng chọn nhiều mục trong hộp danh sách một lần?
26. An Huy muốn xóa tất cả các mục trong hộp danh sách khi người dùng nhấp một nút lệnh đặc biệt, cmdGoAway. Hãy viết thủ tục biến cố cần để xóa hộp danh sách lstVals nếu người dùng nhấp nút lệnh cmdGoAway.
27. Tại sao bạn nên đưa ra một nút lệnh cho người dùng khi thêm các mục vào một điều khiển hộp combo?

Tìm lỗi kỹ thuật

28. An Huy cần định nghĩa một hộp danh sách với 24 mục. Trong thủ tục biến cố Form_Load(), An Huy cố làm điều này:

```
Static lstData(24) ' Define 24 list boxes
```

Hãy cho An Huy biết anh ấy đã làm sai điều gì.

Phần nâng cao

Viết một ứng dụng hỏi người dùng, với một hàm InputBox(), liệu người ấy muốn thấy một hộp combo cuộn xuống, một hộp combo đơn giản, hay một hộp danh sách cuộn xuống. Hỏi người dùng trong chuỗi dấu nhắc InputBox(), việc nhập 1, 2, hay 3 để chỉ loại hộp combo nào trong ba loại hộp combo được yêu cầu. Hãy sử dụng một vòng lặp kiểm tra người dùng sẽ nhập giá trị đúng, và nếu người dùng nhập vào một giá trị ngoài 1, 2, hay 3, giữ dấu nhắc cho đến khi người dùng nhập vào đúng 1 trong 3 giá trị. Sau đó sẽ hiển thị một danh sách 10 mùi kem trong điều khiển hộp combo riêng biệt đó.

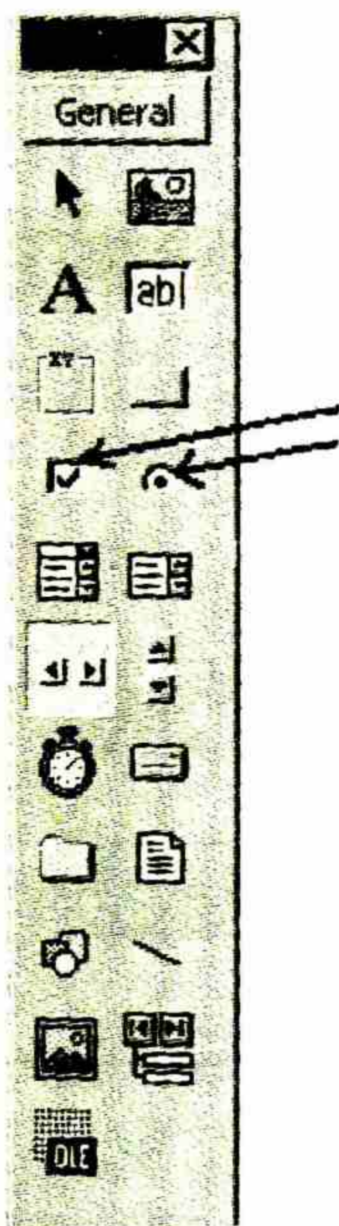
Mách nước

Hãy gán nội dung nhập của người dùng vào thuộc tính Style của hộp combo (Combo Box).

Bài 12

Ô chọn, nút tùy chọn và mảng điều khiển

- **Nút tùy chọn cung cấp chọn lựa**
- **Ô chọn đưa ra nhiều chọn lựa**
- **Nhiều nhóm nút tùy chọn**
- **Các mảng điều khiển đơn giản**



Bạn sẽ thấy rằng bài này không khó bởi vì nó giới thiệu hai điều khiển mới mà bạn đã gặp trong nhiều chương trình Windows, đồng thời xây dựng trên các mảng đã đề cập trong bài trước. Các điều khiển ô chọn (Check Box) và nút tùy chọn (Option Button) mô tả ở đây sẽ giúp người dùng chọn lựa các giá trị dữ liệu và những tùy chọn có thể. Không giống điều khiển hộp danh sách và hộp combo, điều khiển ô chọn và nút tùy chọn (xem Hình 12.1) là những điều khiển hoàn hảo cung cấp cho người dùng một số chọn lựa giới hạn.

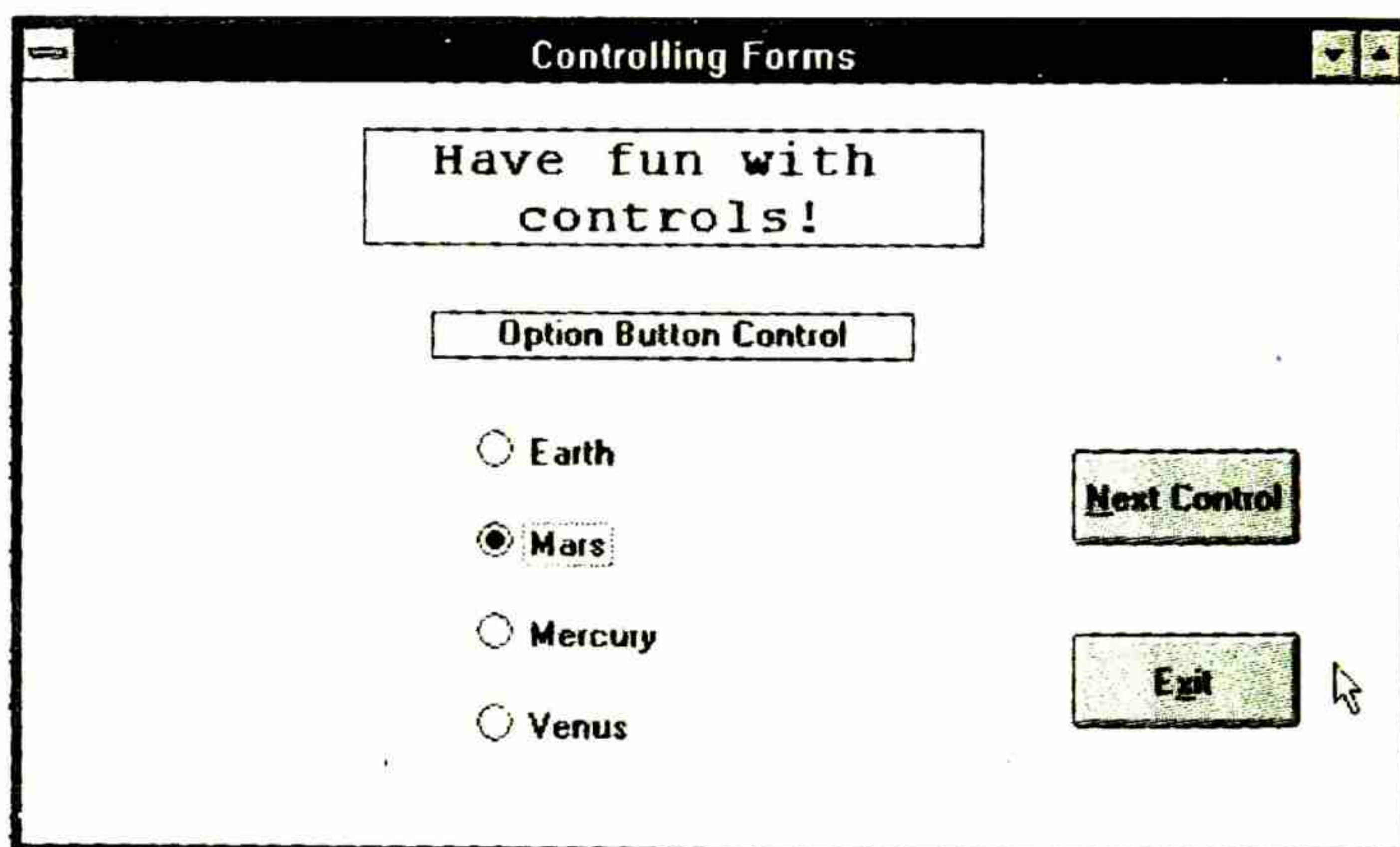
Hình 12.1. *Các điều khiển ô chọn (Check Box) và nút tùy chọn (Option Button).*

NÚT TÙY CHỌN CUNG CẤP CHỌN LỰA

Khái niệm

Điều khiển nút tùy chọn (Option Button) cho người dùng khả năng chọn một khoản mục từ danh sách nhiều mục hiển thị trong dãy tùy chọn option button. Visual Basic cho người dùng chọn nhiều nút tùy chọn một lần. Các biến cố option button cung cấp nhiều cách cho bạn quản lý các chọn lựa nút tùy chọn.

Hộp danh sách và hộp combo là những điều khiển hoàn hảo để hiển thị việc cuộn danh sách các mục mà người dùng chọn và trong trường hợp hộp combo, còn có thể thêm vào các mục. Không giống hộp danh sách và hộp combo, nút tùy chọn sử dụng thích hợp khi bạn phải cung cấp cho người dùng một danh sách những chọn lựa cố định mà chương trình của bạn đã biết trước đó.



Hình 12.2. *Bạn chỉ chọn một nút tùy chọn tại một thời điểm.*

Các nút tùy chọn đôi khi được gọi là *nút đài* (radio button). Có lẽ bạn không nhớ tới radio trong xe hơi có 5 hay 6 nút với các lệnh thiết lập sẵn được gán cho từng nút. Tại thời điểm bất kỳ, bạn có thể ấn một nút bởi vì radio có thể chỉ nghe một đài ở một thời điểm. Khi bạn đã

nhấn nút, bất kỳ một nút đã nhấn nào khác trước đó ngay lập tức sẽ bật lên. Các nút được thiết kế sao cho tại một thời điểm chỉ có thể chọn một nút.

Hãy dành ít thời gian nạp CONTROLS.VBP và nhấn nút lệnh điều khiển Next cho đến khi bạn nhìn thấy các nút tùy chọn trong Hình 12.2.

Bảng 12.1 sẽ liệt kê các xác lập thuộc tính cho điều khiển nút tùy chọn (Option Button) trong cửa sổ Property khi bạn đặt các nút tùy chọn trên mẫu biểu. Có nhiều thuộc tính bạn đã gặp trước đây trong những loại điều khiển khác.

Bảng 12.1. *Các thuộc tính nút tùy chọn*

Thuộc tính	Diễn giải
Alignment	Hoặc bằng 0 để canh trái (mặc định) hoặc bằng 1 để canh phải đề mục (caption) của nút. Nếu bạn chọn canh trái, nút tùy chọn sẽ xuất hiện bên trái caption. Nếu bạn chọn canh phải, nút tùy chọn sẽ xuất hiện bên phải caption.
BackColor	Màu nền của nút tùy chọn. Đây là một số thập lục phân trình bày một trong số hàng ngàn giá trị màu có thể có trong Windows. Bạn có khả năng chọn từ một bảng màu do Visual Basic hiển thị khi ấn định thuộc tính BackColor. Màu nền mặc định chính là màu nền mặc định của mẫu biểu.
Caption	Văn bản sẽ xuất hiện trong một nút tùy chọn. Nếu bạn thiết đặt trước một ký tự bất kỳ trong văn bản với một dấu &, lúc ấy, ký tự đó sẽ hoạt động như một phím truy xuất của nút tùy chọn.
DragIcon	Biểu tượng sẽ xuất hiện khi người dùng kéo nút tùy chọn trên mẫu biểu. (Hiếm khi bạn cho phép người dùng di chuyển một nút tùy chọn, cho nên các xác lập thuộc tính Drag... thường không thích hợp.)
DragMode	Hoặc bằng 1 cho các yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển) hoặc bằng 0 (mặc định) cho việc kéo mouse tự động, nghĩa là người dùng không thể kéo nút tùy chọn (Option Button), mà thông qua mã lệnh, có thể bắt đầu thao tác kéo nếu cần thiết.

Enabled	Nếu định là True (mặc định), nút tùy chọn có thể đáp ứng các biến cố. Ngược lại, Visual Basic dừng việc xử lý biến cố cho điều khiển riêng biệt đó.
FontBold	True (mặc định) tương ứng với việc hiển thị các ký tự Caption dạng chữ đậm; ngược lại là False.
FontItalic	True (mặc định) tương ứng với việc hiển thị các ký tự Caption dạng chữ nghiêng; ngược lại là False.
FontName	Tên kiểu chữ của đề mục (caption) trong nút tùy chọn. Thông thường, bạn sẽ dùng tên một phông chữ TrueType trong Windows.
FontSize	Kích cỡ tính bằng point của phông chữ được sử dụng cho đề mục nút lệnh.
FontStrikethru	True (mặc định) tương ứng việc hiển thị các ký tự Caption dạng gạch ngang (nghĩa là các ký tự có một đường gạch ngang qua nó); ngược lại là False.
FontUnderline	True (mặc định) tương ứng việc hiển thị các ký tự Caption có gạch dưới; ngược lại là False.
ForeColor	Mã màu theo số thập lục phân của màu văn bản đề mục.
Height	Chiều cao tính bằng twip của nút tùy chọn.
HelpContextID	Nếu bạn thêm trợ giúp cảm ngữ cảnh cao cấp vào ứng dụng của bạn, giá trị HelpContextID sẽ cung cấp một số xác định cho văn bản trợ giúp.
Index	Nếu nút tùy chọn (Option Button) là một phần của mảng điều khiển, thuộc tính Index sẽ cung cấp chỉ số cho từng nút tùy chọn riêng biệt.
Left	Khoảng cách tính bằng twip từ biên trái cửa sổ Form tới biên trái nút tùy chọn.
MousePointer	Hình dạng con trỏ mouse nếu người dùng di chuyển con trỏ mouse trên nút tùy chọn. Các giá trị có thể từ 0 đến 12 và trình bày một loạt hình dạng khác nhau mà con trỏ mouse có thể có. (Xem Chương 12.)
Name	Tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên Option1, Option2, và tiếp tục như thế khi bạn thêm lần lượt các nút tùy chọn (Option Button) vào mẫu biểu.

TabIndex	Thứ tự tab tiêu điểm bắt đầu bằng 0 và gia tăng mỗi lần bạn thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi giá trị TabIndex của điều khiển thành những giá trị khác. Không có hai điều khiển nào trên cùng mẫu biểu có cùng giá trị TabIndex.
TabStop	Nếu là True, người dùng có thể nhấn Tab để di chuyển tiêu điểm tới nút tùy chọn này. Nếu là False, nút tùy chọn không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Đây là tiện ích dành cho lập trình viên nhận biết chú thích áp dụng vào nút tùy chọn.
Top	Khoảng cách tính bằng twip từ cạnh trên cùng của một nút tùy chọn tới cạnh trên cùng của mẫu biểu.
Value	True hoặc False (mặc định) cho biết liệu nút tùy chọn được chọn.
Visible	True hoặc False, cho biết liệu người dùng có thể thấy (và vì vậy sử dụng được) nút tùy chọn.
Width	Độ rộng tính bằng twip của nút tùy chọn.

Bảng 12.2 chứa một danh sách biến cố option button kích hoạt các thủ tục biến cố bạn có thể viết. Thông thường, thủ tục biến cố Click là thủ tục phổ biến nhất để một ứng dụng có thể thi hành một hành động xác định khi người dùng chọn một option button riêng biệt.

Bảng 12.2. *Các biến cố option button*

Biến cố	Diễn giải
Click	Xảy ra khi người dùng nhấp nút tùy chọn bằng mouse.
DblClick	Xảy ra khi người dùng nhấp đúp mouse trên nút tùy chọn.
DragDrop	Xảy ra khi một thao tác kéo trên nút tùy chọn hoàn tất.
DragOver	Xảy ra trong suốt quá trình kéo.
GotFocus	Xảy ra khi nút tùy chọn nhận tiêu điểm.
KeyDown	Xảy ra khi người dùng nhấn một phím đồng thời thuộc tính KeyPreview được thiết đặt là True cho điều khiển trên nút tùy chọn; ngược lại, điều khiển lấy biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn một phím trên nút tùy chọn.
KeyUp	Xảy ra khi người dùng thả một phím.
LostFocus	Xảy ra khi nút tùy chọn mất tiêu điểm.

Tóm tắt

Nạp và chạy project TRANSL.VBP. Ví dụ 12.1 chứa mã lệnh các thủ tục biến cố của project. Chương trình cho phép người dùng chọn ngôn ngữ đích là "Good Morning" được dịch. Chỉ có đủ chỗ trên mẫu biểu cho một từ dịch, cho nên những nút tùy chọn đảm bảo rằng người dùng có thể chọn nhiều nhất một từ được dịch.

Ôn lại

Các điều khiển nút tùy chọn (option button) cho phép bạn thêm các tùy chọn vào mẫu biểu. Người dùng được phép chọn nhiều nhất một chọn lựa bằng cách nhấp (hay tính toán để thay đổi tiêu điểm và nhấn phím Enter) nút tùy chọn của tùy chọn mong muốn. Khi thêm mã lệnh vào các thủ tục biến cố Click của nút tùy chọn, bạn có thể đáp ứng lựa chọn của người dùng.

Ví dụ 12.1. *Thủ tục Form_Load() và bốn thủ tục Click () của nút tùy chọn.*

```
Sub
Form_Load ()
' Make sure that all option buttons are
' False when first displaying the form
optFre.Value = 0
optIta.Value = 0
optSpa.Value = 0
optPig.Value = 0
End Sub

Sub cmdExit_Click ()
End
End Sub

Sub optFre_Click ()
' Send the French message
lblTrans.Caption = "Bonjour"
End Sub
```

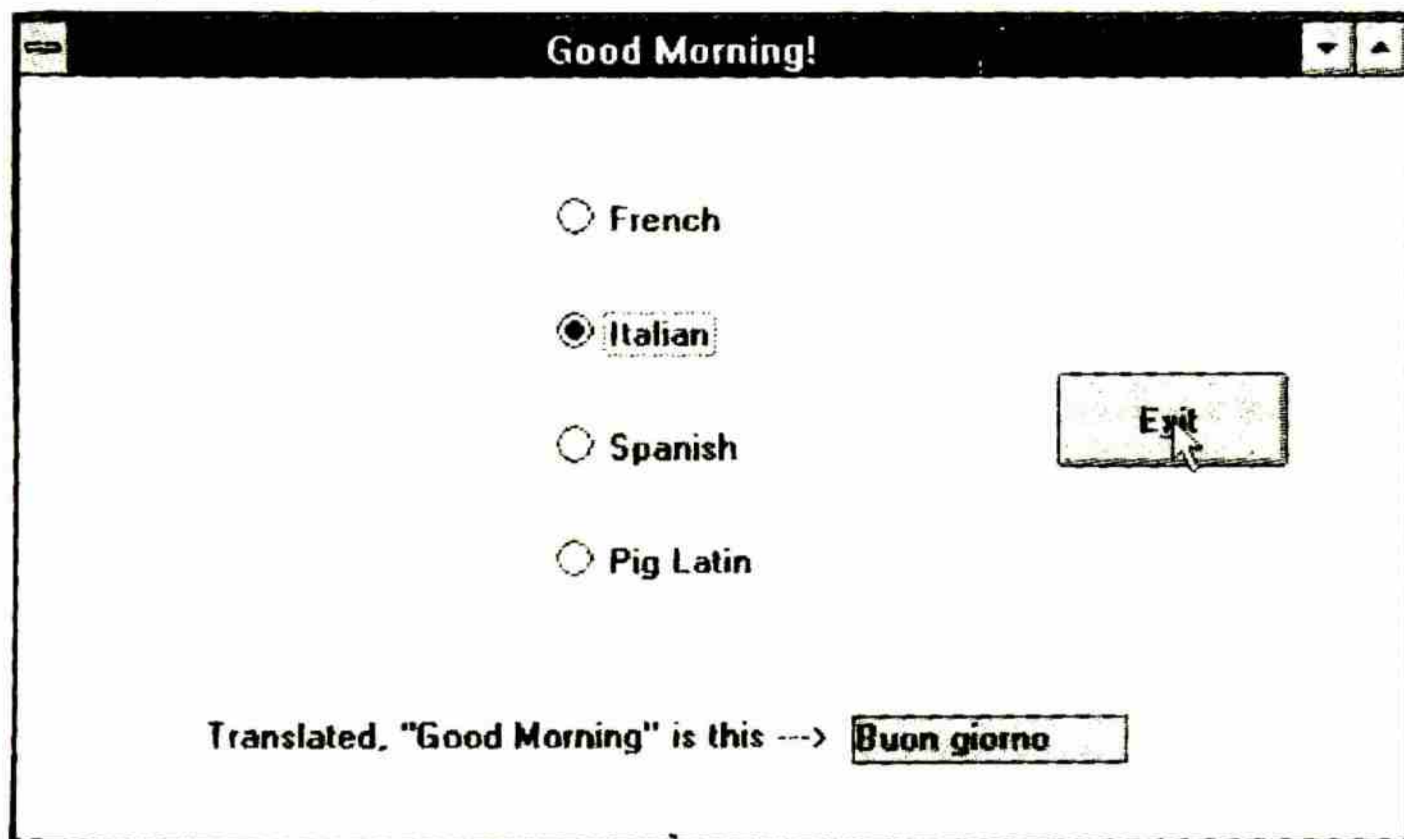
```
Sub optIta_Click ()
' Send the Italian message
lblTrans.Caption = "Buon giorno"
End Sub

Sub optPig_Click ()
' Send the Pig latin message
lblTrans.Caption = "oodGa ayDa"
End Sub

Sub optSpa_Click ()
' Send the Spanish message
lblTrans.Caption = "Buenos đas"
End Sub
```

Kết quả

Hình 12.3 minh họa màn hình TRANSL.VBP sau khi người dùng chọn nút tùy chọn Italian. Việc dịch từ Italian sẽ xuất hiện trên nhãn (label) được tạo bóng ở cuối màn hình.



Hình 12.3. *Người dùng muốn tới parla Italiano bene!*

Phân tích

Ví dụ 12.1 cho thấy một điểm khác biệt nhỏ so với các ví dụ khác mà bạn đã làm việc trước đó trong sách này. Ví dụ có nhiều thủ tục biến cố. Khi bạn tập hợp lại, một chương trình Visual Basic có nhiều thủ tục biến cố. Đến Chương 8, bạn sẽ tìm hiểu về các loại thủ tục khác trong một chương trình Visual Basic.

Mặc dầu thuộc tính Value mặc định cho tất cả các nút tùy chọn là 0, nghĩa là không được chọn, Visual Basic sẽ tự động chọn một nút tùy chọn (nút có giá trị thuộc tính TabIndex thấp nhất) khi nạp ứng dụng. Vì vậy, các dòng 4 đến 7 trong Ví dụ 12.1 sẽ thiết đặt thuộc tính Value của 4 nút tùy chọn bằng 0 trong lúc nạp mẫu biểu để không có nút tùy chọn nào được thiết đặt cho đến khi người dùng chọn 1. (Bốn nút tùy chọn có tên là optFre, optlta, optPig, và optSpa.)

Các dòng 10 đến 12 sẽ cung cấp thủ tục biến cố nút lệnh Exit quen thuộc vốn dừng chương trình khi người dùng nhấn nút Exit.

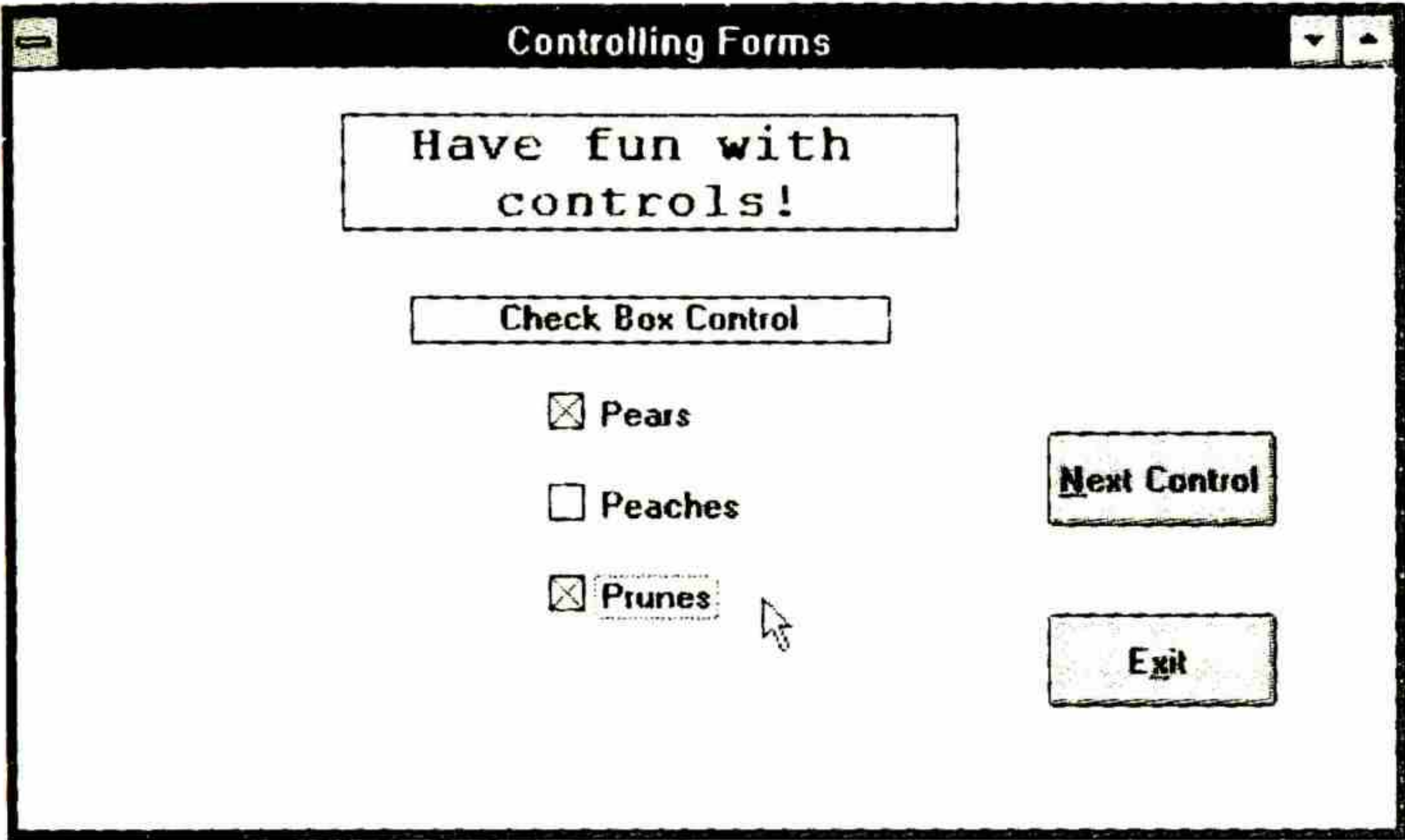
Các dòng 14 đến 32 hoàn thành mã lệnh với 4 thủ tục biến cố Click. Người dùng sẽ kích 1 trong 4 thủ tục bằng việc chọn 1 trong 4 nút tùy chọn. Label dịch nghĩa được tạo bóng ở cuối màn hình có tên là lblTrans. Bốn thủ tục biến cố Click sẽ thay đổi thuộc tính đề mục Caption của đối tượng lblTrans thành bản dịch phù hợp với đề mục của nút tùy chọn.

Ô CHỌN ĐƯA RA NHIỀU CHỌN LỰA

Khái niệm

Điều khiển ô chọn (Check Box) làm việc y như điều khiển nút tùy chọn (Option Button). Bạn sẽ đặt nhiều ô chọn (Check Box) trên một mẫu biểu; mỗi ô chọn trình bày một tùy chọn do người dùng chọn. Không giống nút tùy chọn, người dùng có thể chọn nhiều ô chọn (Check Box) tại một thời điểm.

Hình 12.4 có các điều khiển ô chọn đã tìm thấy trong ứng dụng CONTROLS.VBP của sách này. Nếu bạn nạp và chạy chương trình, các ô chọn là bốn bộ điều khiển bạn sẽ nhấp trên đó. Qua Hình 12.4, khi người dùng chọn một ô chọn, ô chọn đó sẽ được đánh dấu X, cho biết một chọn lựa thật sự. Hai trong ba ô chọn (Check Box) được chọn trong Hình 12.4.



Hình 12.4. *Các điều khiển ô chọn cho phép nhiều chọn lựa.*

Ghi chú

Mặc dầu các điều khiển ô chọn trong Hình 12.4 không chứa phím truy xuất, bạn vẫn có thể thêm phím truy xuất vào các điều khiển ô chọn y như thêm chúng vào các nút tùy chọn vậy.

Bảng 12.3 chứa các giá trị thuộc tính có hiệu lực cho điều khiển ô chọn. Đa số thuộc tính tương đương với các thuộc tính nút tùy chọn.

Bảng 12.3. *Các thuộc tính ô chọn*

Thuộc tính	Diễn giải
Alignment	Hoặc bằng 0 để canh trái (mặc định) hoặc bằng 1 để canh phải đề mục (caption) của ô chọn. Nếu bạn chọn canh trái, ô chọn sẽ xuất hiện bên trái caption. Nếu bạn chọn canh phải, ô chọn sẽ xuất hiện bên phải caption.
BackColor	Màu nền của ô chọn (Check Box). Đây là một số thập lục phân trình bày một trong số hàng ngàn giá trị màu có thể có trong Windows. Bạn có khả năng chọn từ một bảng màu do Visual Basic hiển thị khi ấn định thuộc tính BackColor. Màu nền mặc định chính là màu nền mặc định của mẫu biểu.

Caption	Văn bản sẽ xuất hiện trong một ô chọn. Nếu bạn thiết đặt trước một ký tự bất kỳ trong văn bản một dấu &, lúc ấy, ký tự đó sẽ hoạt động như một phím truy xuất của ô chọn.
DragIcon	Biểu tượng sẽ xuất hiện khi người dùng kéo ô chọn (Check Box) trên mẫu biểu. (Hiếm khi bạn cho phép người dùng di chuyển một ô chọn, cho nên các xác lập thuộc tính Drag... thường không thích hợp).
DragMode	Hoặc bằng 1 cho các yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển) hoặc bằng 0 (mặc định) cho việc kéo mouse tự động, nghĩa là người dùng không thể kéo ô chọn, mà thông qua mã lệnh, có thể bắt đầu thao tác kéo nếu cần thiết.
Enabled	Nếu định là True (mặc định), ô chọn có thể đáp ứng các biến cố. Ngược lại, Visual Basic dừng việc xử lý biến cố cho điều khiển riêng biệt đó.
FontBold	True (mặc định) tương ứng với việc hiển thị các ký tự Caption dạng chữ đậm; ngược lại là False.
FontItalic	True (mặc định) tương ứng với việc hiển thị các ký tự Caption dạng chữ nghiêng; ngược lại là False.
FontName	Tên kiểu chữ của đề mục (caption) trong ô chọn. Thông thường, bạn sẽ dùng tên một phông chữ TrueType trong Windows.
FontSize	Kích cỡ tính bằng point của phông chữ được sử dụng cho đề mục của nút lệnh.
FontStrikethru	True (mặc định) tương ứng việc hiển thị các ký tự Caption dạng gạch ngang (nghĩa là các ký tự có một đường gạch ngang qua nó); ngược lại là False.
FontUnderline	True (mặc định) tương ứng việc hiển thị các ký tự Caption có gạch dưới; ngược lại là False.
ForeColor	Mã màu theo số thập lục phân của màu văn bản đề mục.
Height	Chiều cao tính bằng twip của ô chọn.
HelpContextID	Nếu bạn thêm trợ giúp cảm ngữ cảnh cao cấp vào ứng dụng của bạn, giá trị HelpContextID sẽ cung cấp một số xác định cho văn bản trợ giúp.

Index	Nếu ô chọn là một phần của một mảng điều khiển, thuộc tính Index sẽ cung cấp chỉ số cho từng ô chọn riêng biệt.
Left	Khoảng cách tính bằng twip từ biên trái cửa sổ Form tới biên trái ô chọn.
MousePointer	Hình dạng con trỏ mouse nếu người dùng di chuyển con trỏ mouse trên ô chọn. Các giá trị có thể từ 0 đến 12 và trình bày một loạt hình dạng khác nhau mà con trỏ mouse có thể có. (Xem Chương 12.)
Name	Tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên Check1, Check2, và tiếp tục như thế khi bạn thêm lần lượt các ô chọn vào mẫu biểu.
TabIndex	Thứ tự tab tiêu điểm bắt đầu bằng 0 và gia tăng mỗi lần bạn thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi giá trị TabIndex của điều khiển thành những giá trị khác. Không có 2 điều khiển nào trên cùng mẫu biểu có cùng giá trị TabIndex.
TabStop	Nếu là True, người dùng có thể nhấn Tab để di chuyển tiêu điểm tới ô chọn này. Nếu là False, ô chọn không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Đây là tiện ích dành cho lập trình viên nhận biết chú thích áp dụng vào ô chọn.
Top	Khoảng cách tính bằng twip từ đỉnh của một ô chọn tới đỉnh của mẫu biểu.
Value	Hoặc là 0–Unchecked (mặc định), 1–Checked, hay 2–Grayed, cho biết liệu ô chọn được chọn.
Visible	True hoặc False, cho biết liệu người dùng có thể thấy (và vì vậy sử dụng được) ô chọn.
Width	Độ rộng tính bằng twip của ô chọn.

Bảng 12.4 chứa các biến cố có hiệu lực cho điều khiển ô chọn (Check Box).

Bảng 12.4. *Các biến cố ô chọn*

Biến cố	Diễn giải
Click	Xảy ra khi người dùng nhấp ô chọn bằng mouse.
DragDrop	Xảy ra khi một thao tác kéo trên ô chọn hoàn thành.
DragOver	Xảy ra trong suốt quá trình kéo.
GotFocus	Xảy ra khi ô chọn nhận tiêu điểm.
KeyDown	Xảy ra khi người dùng nhấn một phím đồng thời thuộc tính KeyPreview được thiết đặt là True cho các điều khiển trên ô chọn; ngược lại, điều khiển nhận biến cố KeyDown.
KeyPress	Xảy ra khi người dùng nhấn một phím trên ô chọn.
KeyUp	Xảy ra khi người dùng nhả một phím.
LostFocus	Xảy ra khi ô chọn mất tiêu điểm.

Tóm tắt

Hãy nạp và chạy project CHECK.VBP. Ví dụ 12.2 có mã lệnh về các thủ tục biến cố của project. Chương trình cho phép người dùng chọn một trong số bốn tùy chọn định dạng đặc biệt cho bài thơ trong một label được viền. Bài thơ có thể chấp nhận bất kỳ hay tất cả bốn dạng tùy chọn định dạng, cho nên các điều khiển ô chọn cho phép người dùng chọn một hay nhiều tùy chọn định dạng.

Ôn lại

Các điều khiển ô chọn (Check Box) cho phép người dùng chọn nhiều chọn lựa tại một thời điểm, không giống các điều khiển nút tùy chọn (Option Button), chỉ cho phép chọn một tùy chọn (option) tại một thời điểm.

Ví dụ 12.2. *Mã lệnh trong project CHECK.VBP.*

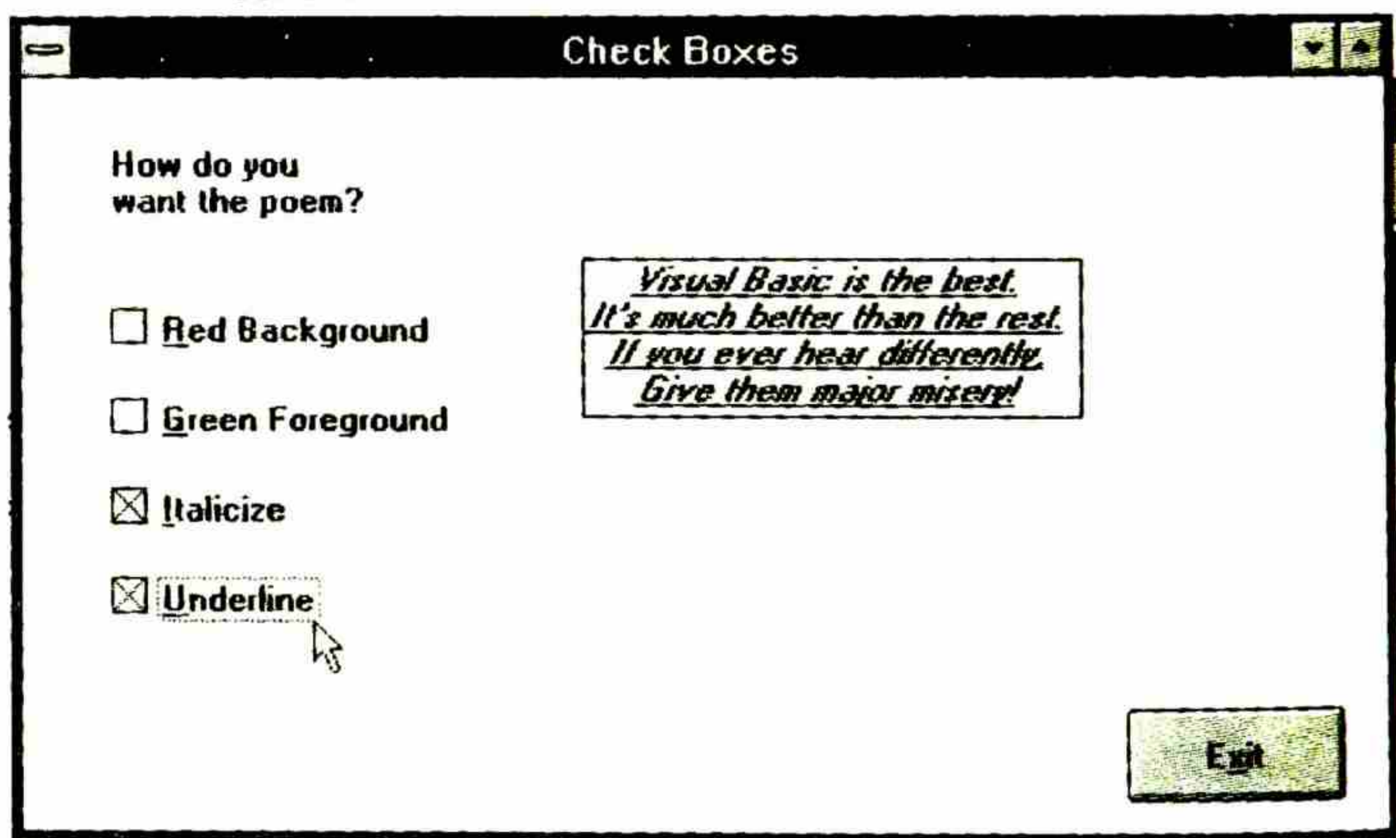
```
Sub Form_Load ()
' Fill the label with a poem.
' The poem will display initially
' using the default label properties.
Dim Newline As String
' The newline sends a carriage return and
```

```
' line feed sequence to the poem so that
' the poem can appear as a multiple-line
' poem.
Newline = Chr$(13) + Chr$(10)
' Fill the label with the poem
' and concatenate the newline characters
lblPoem = "Visual Basic is the best."
lblPoem = lblPoem & Newline
lblPoem = lblPoem & "It's much better than the rest."
lblPoem = lblPoem & Newline
lblPoem = lblPoem & "If you ever hear differently,"
lblPoem = lblPoem & Newline
lblPoem = lblPoem & "Give them major misery!"
End Sub
Sub chkBack_Click ()
' Set the poem label's background to Red or white
If (lblPoem.BackColor = RED) Then
lblPoem.BackColor = WHITE ' Default
Else
lblPoem.BackColor = RED
End If
End Sub
Sub chkFore_Click ()
' Set the poem label's foreground to Green or Black
If (lblPoem.ForeColor = GREEN) Then
lblPoem.ForeColor = BLACK ' Default
Else
lblPoem.ForeColor = GREEN
End If
End Sub
Sub chkItal_Click ()
' Set the poem label's caption to italicize or not
```

```
If (lblPoem.FontItalic = True) Then
lblPoem.FontItalic = False ' Default
Else
lblPoem.FontItalic = True
End If
End Sub

Sub chkUnd_Click ()
' Set the poem label's caption to underline or not
If (lblPoem.FontUnderline = True) Then
lblPoem.FontUnderline = False ' Default
Else
lblPoem.FontUnderline = True
End If
End Sub

Sub cmdExit_Click ()
End
End Sub
```



Hình 12.5. *Chọn hai ô chọn cùng lúc.*

Hình 12.5 mô tả hình ảnh bài thơ khi người dùng chọn hai tùy chọn check box.

Phân tích

Các dòng từ 1 đến 20 có thủ tục biến cố Form_Load() dài nhất. Mục đích của thủ tục là để khởi tạo điều khiển nhãn (Label) với nhiều dòng thơ. Không có thuộc tính MultiLine cho nhãn, cho nên bạn phải đánh lừa Visual Basic bằng cách hiển thị nhiều dòng trong nhãn.

Khái niệm mới

Ký tự điều khiển (control character) ***đưa ra một hành động, không phải một ký tự văn bản.***

Hàm Chr$() (được gọi là hàm chuỗi ký tự hay đôi khi gọi là hàm ký tự) sẽ chấp nhận một số trong cặp ngoặc đơn của nó. Số phải đến từ bảng mã ASCII. Visual Basic đổi số đó thành ký tự ASCII tương ứng của nó. Số ASCII 13 là *ký tự trở về đầu dòng* (carriage return) và là ký tự điều khiển đặc biệt gửi con trỏ về đầu dòng. Số ASCII 10 là *ký tự xuống dòng* (line feed) gửi con trỏ tới dòng kế tiếp trên màn hình. Kết quả của việc kết hợp giá trị ASCII 13 và 10 (dòng 10) là hình thành một ký tự điều khiển đôi đặc biệt.

Ghi chú

Thuộc tính Alignment của nhãn bài thơ được thiết đặt là 2–Center, cho nên các dòng bên trong nhận luôn luôn hiển thị ở giữa đường viền nhãn.

Các dòng từ 13 đến 19 sau đó sẽ xây dựng nhiều dòng thơ bằng cách nối dòng với biến chuỗi kết hợp dòng mới vào thuộc tính Caption của nhãn (label).

Ghi chú

Các dòng từ 13 đến 19 thậm chí không quan tâm thuộc tính Caption! Nó sẽ hiện nội dung bài thơ đang được gửi tới nhãn (label) mà không có bất kỳ thuộc tính nào. Nó xem mỗi điều khiển có một thuộc tính mặc định, trừ khi bạn xác định tên thuộc tính khác, hoạt động như thuộc tính mặc định. Vì vậy, Visual Basic sẽ gán tất cả dòng bài thơ vào thuộc tính Caption trong các dòng 13 đến 19 bởi vì thuộc tính Caption là thuộc tính mặc định cho các nhãn.

Các dòng 22 đến 56 có 4 thủ tục biến cố Click của ô chọn. Lệnh If–Else đầu tiên sẽ kiểm tra xem liệu thuộc tính của bài thơ có định ở giá trị được chọn. Nếu như thế, phần True của lệnh If sẽ định thuộc tính trở lại tình trạng mặc định của nó. Nếu thuộc tính đã chứa giá trị mặc định, lệnh Else thiết đặt thuộc tính thành trạng thái mới.

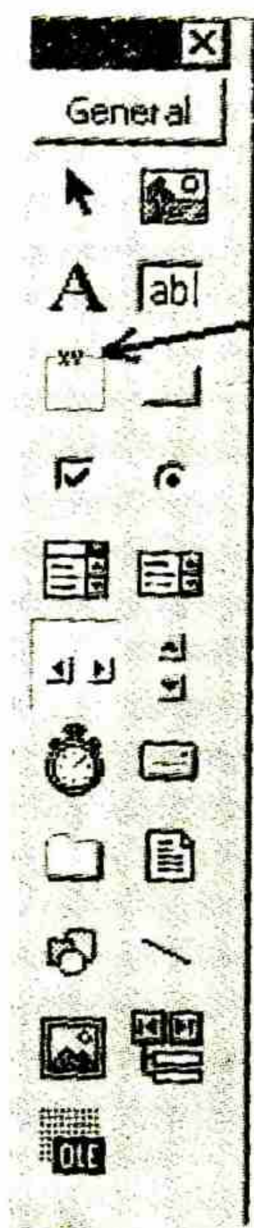
Hầu như ứng dụng luôn có một thủ tục biến cố nút lệnh Exit trong các dòng 58 đến 60, sẽ dừng chương trình theo yêu cầu người dùng.

NHIỀU NHÓM NÚT TÙY CHỌN

Khái niệm

Người dùng có thể chỉ chọn một nút tùy chọn trong số nhiều nút tùy chọn trên mẫu biểu. Có một cách đặt nhiều nhóm nút tùy chọn trên một mẫu biểu bên trong *khung* (frame). Người dùng dễ dàng chọn một nút tùy chọn trong một khung (frame) bất kỳ tại một thời điểm.

Hình 12.6 cho thấy vị trí điều khiển khung (frame) hiện diện trên cửa sổ Toolbox. Điều khiển khung là điều khiển nắm giữ các điều khiển khác. Nếu đặt nhiều khung trên mẫu biểu, bạn có thể nhóm nhiều bộ nút tùy chọn trên mẫu biểu với nhau.



Khi bạn muốn đặt nhiều bộ nút tùy chọn trên một mẫu biểu, hãy chắc chắn đặt nhiều khung trên mẫu biểu đầu tiên. Khung (frame) phải đủ lớn để giữ nhiều nút tùy chọn theo từng nhóm yêu cầu.

Hãy cẩn thận về việc đặt các nút tùy chọn khi bạn muốn điều chỉnh chúng trong các điều khiển khung. Bạn phải vẽ các nút tùy chọn bên trong khung. Bạn không thể tạo các nút tùy chọn bất cứ nơi đâu và chuyển chúng vào khung. Trong đa số ứng dụng, bạn có thể nhấp đúp các điều khiển để đặt chúng vào giữa mẫu biểu, và sau đó kéo điều khiển tới vị trí cuối cùng của nó và định lại kích cỡ cho chúng. Khi đặt các nút tùy chọn bên trong khung, bạn phải nhấp (không nhấp đúp) điều khiển nút tùy chọn trên cửa sổ Toolbox và sau đó vẽ nút tùy chọn bằng việc nhấp mouse bên trong khung và kéo mouse cho đến khi bạn đạt xấp xỉ kích thước nút tùy chọn. Khi bạn nhả nút mouse, Visual Basic sẽ vẽ nút tùy chọn theo kích thước bạn đã vẽ nó.

Hình 12.6. *Vị trí của điều khiển Khung.*

Tóm tắt

Hãy nạp và chạy project OPTIONS.VBP. Ví dụ 12.3 có mã lệnh cho các thủ tục biến cố của project. Chương trình tương tự ứng dụng CHECK.VBP trong phần trước. Các nhóm nút tùy chọn sẽ bật/tắt những thuộc tính định dạng bài thơ đã chọn.

Ôn lại

Khung (frame) cho phép bạn nhóm các điều khiển nút tùy chọn với nhau. Phải đặt khung trước khi vẽ các điều khiển nút tùy chọn bên trong khung.

Ví dụ 12.3. *Mã lệnh cho các nút tùy chọn được nhóm trong khung.*

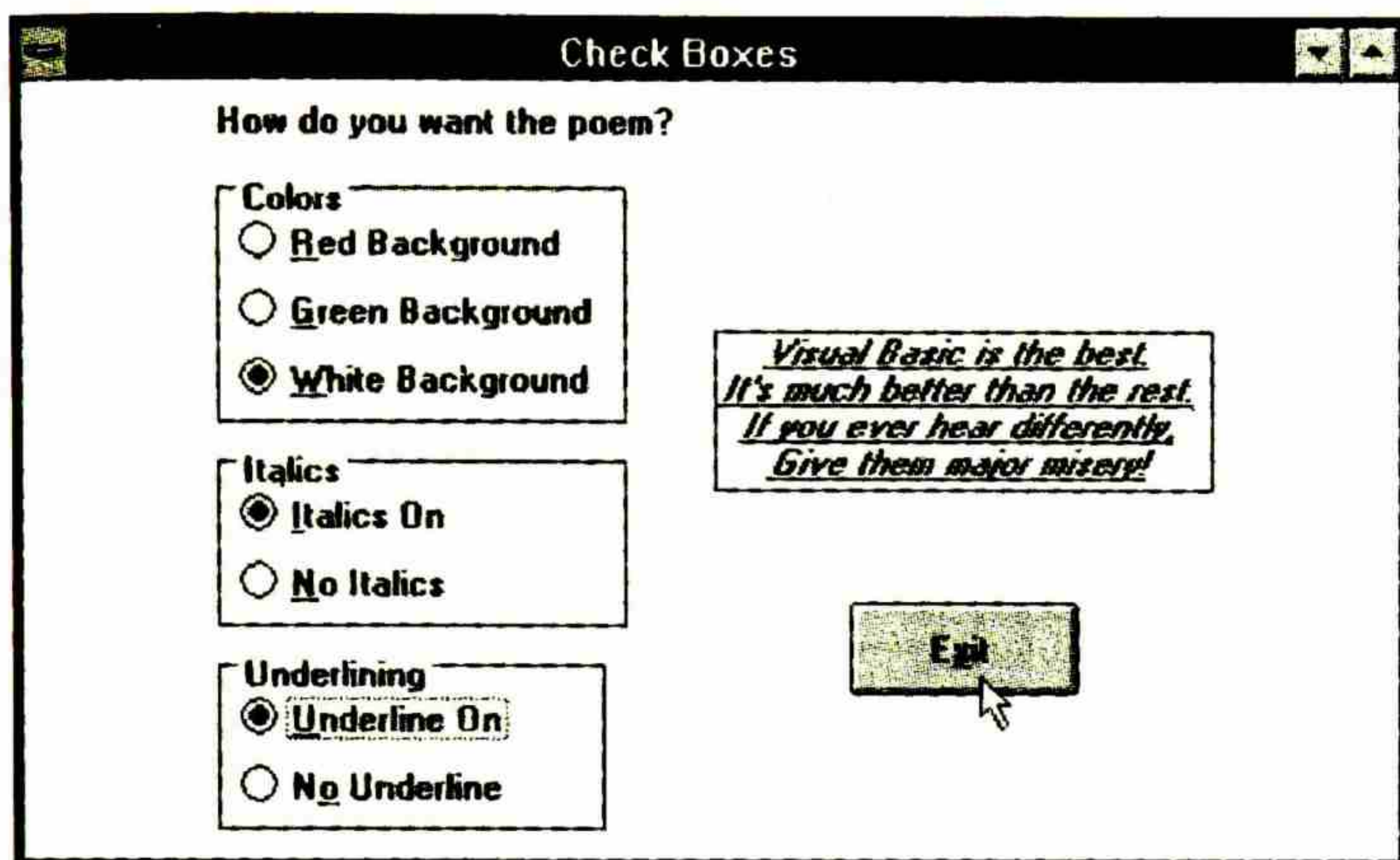
```
Sub Form_Load ()
' Fill the label with a poem.
' The poem will display initially
' using the default label properties.
Dim Newline As String
' The newline sends a carriage return and
' line feed sequence to the poem so that
' the poem can appear as a multiple–line
' poem.
Newline = Chr$(13) + Chr$(10)
' Fill the label with the poem
' and concatenate the newline characters
lblPoem = "Visual Basic is the best."
lblPoem = lblPoem & Newline
lblPoem = lblPoem & "It's much better than the rest."
lblPoem = lblPoem & Newline
lblPoem = lblPoem & "If you ever hear differently,"
lblPoem = lblPoem & Newline
lblPoem = lblPoem & "Give them major misery!"
End Sub

Sub optGreen_Click ()
```

```
lblPoem.BackColor = GREEN
End Sub
Sub optItal_Click ()
lblPoem.FontItalic = True
End Sub
Sub optNoItal_Click ()
lblPoem.FontItalic = False
End Sub
Sub optNoUnd_Click ()
lblPoem.FontUnderline = False
End Sub
Sub optRed_Click ()
lblPoem.BackColor = RED
End Sub
Sub optUnd_Click ()
lblPoem.FontUnderline = True
End Sub
Sub optWhite_Click ()
lblPoem.BackColor = WHITE
End Sub
Sub cmdExit_Click ()
End
End Sub
```

Kết quả

Hình 12.7 cho thấy ứng dụng OPTIONS.VBP sau khi người dùng chọn một tùy chọn trong một khung với cả ba khung.



Hình 12.7. *Các nút tùy chọn đưa vào khung chấp nhận nhiều xác lập.*

Phân tích

Khó khăn của các điều khiển nút tùy chọn khi đưa vào khung là việc đặt các nút tùy chọn bên trong khung, mặc dù điều đó không phải là không làm được. Sau khi bạn vẽ các nút tùy chọn và khung, những thủ tục biến cố thông thường bạn đang viết sẽ làm việc với các điều khiển.

Từ dòng 22 đến 48 bao gồm các thủ tục biến cố Click vốn thiết đặt những thuộc tính định dạng nhãn (label) của bài thơ.

CÁC MẢNG ĐIỀU KHIỂN ĐƠN GIẢN

Khái niệm

Ưu điểm khi đặt một hay nhiều điều khiển giống nhau bên trong một mảng điều khiển, bạn có thể viết một thủ tục biến cố đơn nhưng lại xử lý được nhiều điều khiển trên mẫu biểu.

Thuật ngữ mới

Mảng điều khiển là một danh sách các điều khiển có cùng tên.

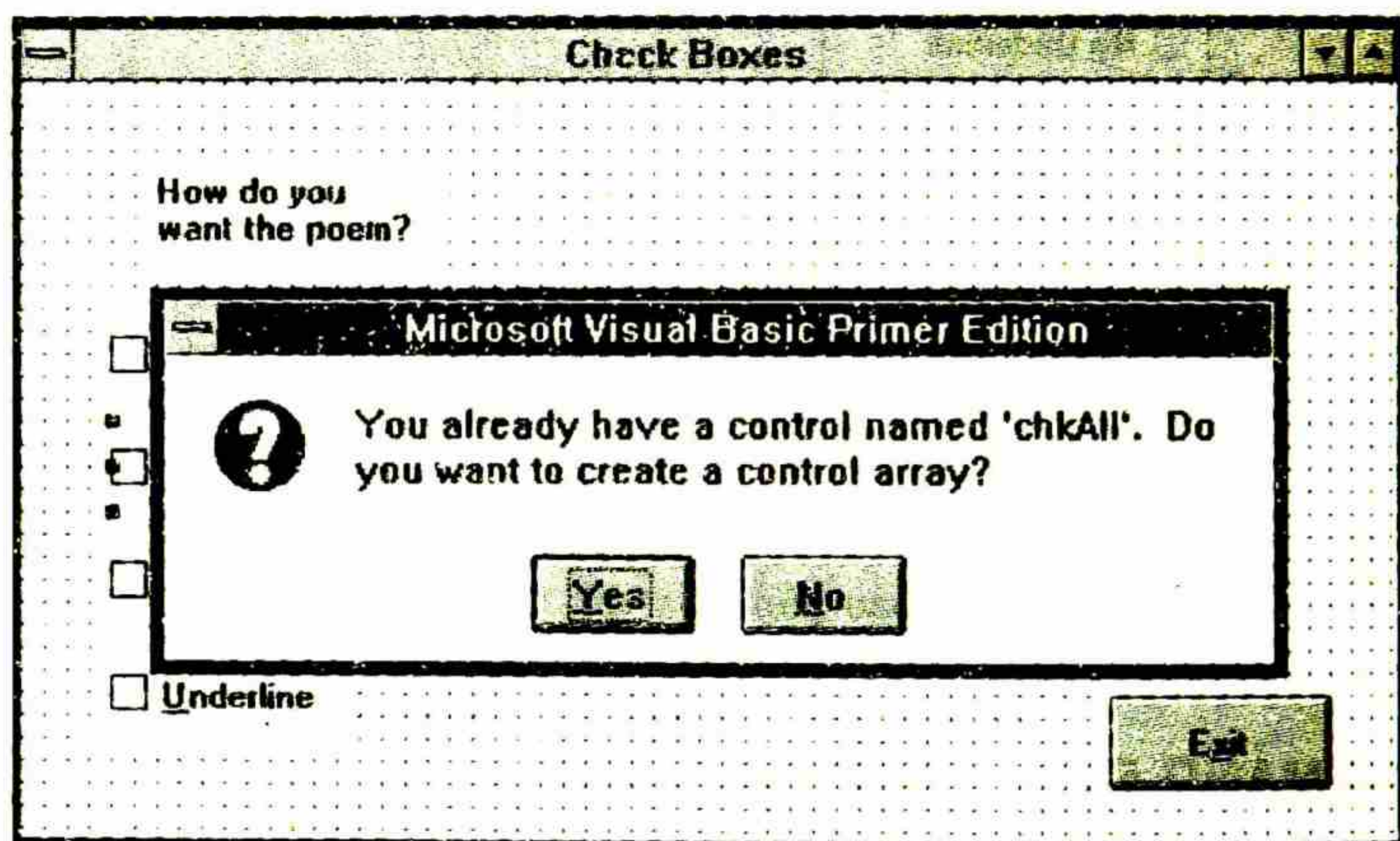
Khi bạn đặt nhiều điều khiển có cùng kiểu điều khiển trên một mẫu biểu và gán cùng thuộc tính Name tới các điều khiển này, có nghĩa

bạn đang tạo một mảng điều khiển. Như với biến mảng, một chỉ số bắt đầu bằng 0 sẽ phân biệt một điều khiển với các điều khiển khác. Thuộc tính Index của mỗi điều khiển trong mảng bao gồm vị trí chỉ số của điều khiển trong mảng. Nếu bạn muốn đánh số lại các điều khiển trong mảng bắt đầu là 1, bạn có thể làm như thế bằng cách thay đổi giá trị Index của điều khiển có giá trị index bằng 0 đầu tiên thành 1.

Ghi chú

Mỗi điều khiển trong một mảng điều khiển phải cùng kiểu dữ liệu.

Nếu có 5 hộp nhập (Text Box) trong một mảng điều khiển có tên là txtBoxes, mỗi điều khiển riêng biệt sẽ được đặt tên là txtBoxes(0), txtBoxes(1), txtBoxes(2), txtBoxes(3), và txtBoxes(4).



Hình 12.8. *Visual Basic bảo đảm rằng bạn muốn một mảng điều khiển.*

Ngay khi bạn đặt một điều khiển trên mẫu biểu có cùng tên như một điều khiển đang tồn tại, Visual Basic chắc chắn rằng bạn muốn bắt đầu một mảng điều khiển bằng cách đưa ra các hộp thông báo (Message Box) như trong Hình 12.8. Những nhà thiết kế Visual Basic biết rằng có thể bạn sẽ tình cờ thử đặt hai điều khiển trên cùng mẫu biểu có cùng tên, và hộp thông báo dùng để khẳng định ý định tạo một mảng điều khiển. Nếu bạn chọn nút lệnh No trong hộp thông báo cảnh báo, Visual Basic sẽ trả lại điều khiển thứ hai một tên mặc định của nó.

Tóm tắt

Hãy nạp và chạy project Check2.VBP. Ví dụ 12.4 chứa mã lệnh thủ tục biến cố chkAll_Click() của project. Chương trình tương tự ứng dụng CHECK.VBP bạn đã làm quen trong phần trước, có khác chăng chỉ là thay vì phân biệt 4 thủ tục biến cố ô chọn, CHECK2.VBP chứa một mảng điều khiển ô chọn đơn có tên là chkAll.

Ôn lại

Bằng cách đặt nhiều điều khiển trong một mảng điều khiển, bạn có thể tham khảo từng điều khiển bằng chỉ số của nó thay vì một tên khác nhau cho từng điều khiển. Không cần viết nhiều thủ tục biến cố, bạn chỉ cần viết một thủ tục biến cố đơn.

Ví dụ 12.4. *Mã lệnh thủ tục biến cố của một mảng điều khiển.*

```
Sub chkAll_Click (Index As Integer)
' A single Select Case
' handles all check box formats
Select Case Index
Case 0:
' Set the poem label's background to Red or white
If (lblPoem.BackColor = RED) Then
lblPoem.BackColor = WHITE ' Default
Else
lblPoem.BackColor = RED
End If
Case 1:
' Set the poem label's foreground to Green or Black
If (lblPoem.ForeColor = GREEN) Then
lblPoem.ForeColor = BLACK ' Default
Else
lblPoem.ForeColor = GREEN
End If
Case 2:
' Set the poem label's caption to italicize or not
If (lblPoem.FontItalic = True) Then
```

```
22: lblPoem.FontItalic = False ' Default
23: Else
24: lblPoem.FontItalic = True
25: End If
26: Case 3:
27: ' Set the poem label's caption to underline or not
28: If (lblPoem.FontUnderline = True) Then
29: lblPoem.FontUnderline = False ' Default
30: Else
31: lblPoem.FontUnderline = True
32: End If
33: End Select
34: End Sub
```

Phân tích

Lệnh Select Case xử lý tất cả bốn ô chọn. Chỉ có một thủ tục Click() của ô chọn trong toàn bộ ứng dụng dù cho có bốn ô chọn. Mỗi ô chọn có tên là chkAll, cho nên chúng là một mảng điều khiển các ô chọn và chỉ có một thủ tục biến cố cho từng biến cố mà lập trình viên muốn lập trình.

Vì vậy, bất cứ khi nào người dùng nhấp ô chọn (Check Box) bất kỳ, thủ tục biến cố chkAll_Click() sẽ thi hành. Làm cách nào thủ tục biết ô chọn nào đã kích hoạt biến cố? Dòng 1, dòng đầu tiên của thủ tục biến cố, cho bạn một đầu mối. Visual Basic sẽ gửi giá trị Index của ô chọn mà người dùng đã nhấp vào thủ tục biến cố. Cặp dấu ngoặc đơn là một cơ chế mà Visual Basic sử dụng để chọn các giá trị như là index trong mảng điều khiển đã kích hoạt biến cố. Bạn sẽ tìm hiểu các giá trị ngoặc đơn như thế trong Chương 8, nhưng cho đến lúc đó, hãy đảm bảo rằng sau khi thủ tục chkAll_Click() thi hành, một biến Index sẽ chứa chỉ số của ô chọn trong mảng điều khiển mà người dùng đã chọn.

Lệnh Select Case sẽ thống nhất các thủ tục biến cố Click() của nhiều ô chọn trong Hình 12.2. Mảng điều khiển sẽ tiết kiệm thời gian và bộ nhớ bởi vì một thủ tục biến cố đơn bây giờ có thể xử lý tất cả biến cố trong mảng điều khiển.

Bài tập

Kiến thức tổng quát

1. Điều khiển nào cung cấp chọn lựa đơn từ một tập hợp nhiều chọn lựa?
2. Điều khiển nào tốt nhất cho phép người dùng chọn một hay nhiều tùy chọn?
3. Tại sao các nút tùy chọn (Option Button) còn được gọi là nút đài(radio button)?
4. Điều gì sẽ xảy ra nếu người dùng chọn một nút tùy chọn khác với nút tùy chọn hiện hành được chọn?
5. Đúng hay Sai: Bạn có thể viết mã lệnh đảm bảo rằng không có nút tùy chọn nào được chọn khi chương trình bắt đầu thi hành.
6. Thuộc tính Alignment làm gì với nút tùy chọn (Option Button) và ô chọn (Check Box)?
7. Thuộc tính nào xác định một nút tùy chọn được chọn?
8. Thuộc tính nào xác định một ô chọn được chọn?
9. Đúng hay Sai: Một hay nhiều thuộc tính Value có thể được thiết đặt cho các nút tùy chọn trên một mẫu biểu (giả sử rằng nút tùy chọn không được đặt trong khung).
10. Đúng hay Sai: Một hay nhiều thuộc tính Value có thể đặt cho các nút tùy chọn trên một mẫu biểu (giả sử rằng các nút tùy chọn được đặt trong khung).
11. Khung (frame) là gì?
12. Đúng hay Sai: Bạn có thể đặt nhiều khung trên một mẫu biểu tại thời điểm bất kỳ.
13. Ký tự điều khiển là gì?
14. Hãy mô tả cách khởi tạo một điều khiển nhãn (Label) với nhiều dòng văn bản.
15. Hàm Chr$() là gì?
16. Mảng điều khiển là gì?
17. Nếu một mảng điều khiển có bảy điều khiển, có bao nhiêu tên mà các điều khiển có?

18. Bằng cách nào bạn phân biệt giữa các giá trị trong một mảng điều khiển?
19. Chỉ số bắt đầu mặc định cho các mảng điều khiển là gì?
20. Có bao nhiêu thủ tục biến cố bạn phải viết cho một mảng điều khiển có tám điều khiển?
21. Thuộc tính giữ chỉ số của một mảng điều khiển là gì?
22. Cách Visual Basic biết bạn muốn tạo một mảng điều khiển?

Viết mã lệnh...

23. Hãy viết dòng lệnh mở một thủ tục biến cố DblClick cho mảng điều khiển nút lệnh có tên là cmdAll.

Tìm lỗi kỹ thuật

24. An Huy đang thử điều khiển những khung và nút tùy chọn. Đầu tiên, An Huy đặt ba khung trên mẫu biểu của anh ấy. Sau đó, An Huy nhấp đúp điều khiển nút tùy chọn (Option Button) trên cửa sổ Toolbox 9 lần và di chuyển từng điều khiển tới khung riêng của nó. An Huy dường như không thể chọn nhiều hơn một nút tùy chọn tại một thời điểm trên mẫu biểu, ngay cả khi ba tập hợp nút tùy chọn xuất hiện trên các khung khác nhau. An Huy đang làm sai điều gì?

Phần nâng cao

Ký tự nào trong câu lệnh gán sau đặt vào biến chuỗi có tên

MyGrade?

```
MyGrade = Chr$(65)
```

Hãy tạo một mẫu biểu chứa 10 nút tùy chọn, gồm hai bộ trong hai khung, mỗi bộ 5 nút tùy chọn. Hãy đặt 10 nút tùy chọn trong hai mảng điều khiển. Nhãn cho mỗi nút tùy chọn là One, Two, Three, Four, và Five. Hãy viết hai thủ tục biến cố Click cho từng mảng điều khiển nút tùy chọn, phát tiếng bíp ra loa PC một khi bất kỳ nút tùy chọn nào được chọn vào lúc đó. Nói cách khác, nếu người dùng nhấp nút tùy chọn Three của khung bên trái, và nút tùy chọn có giá trị Two ở khung bên phải, bạn sẽ nhấp loa 5 lần.

Bài thực hành 6

Kết hợp mã lệnh và điều khiển

Tóm tắt

Chương này chỉ bạn cách làm việc với dữ liệu mảng nâng cao cũng như cách thiết đặt điều khiển nút tùy chọn (Option Button) và ô chọn (Check Box) trên mẫu biểu. Các mảng làm đơn giản quá trình lập trình bằng cách giúp bạn duyệt qua dữ liệu và điều khiển với vòng lặp For. Nút tùy chọn và ô chọn cho người dùng nhiều cách lựa chọn.

Bây giờ bạn có thể bổ sung điều khiển vào chương trình của bạn bằng cách đặt các vòng lặp trong mã lệnh để lặp lại một phần chương trình. Máy tính là công cụ tuyệt vời cho quá trình tính lặp nhiều lần để có được kết quả. Vòng lặp cũng cung cấp cho bạn cách kiểm tra giá trị dữ liệu nhập vào của người dùng là Đúng hay Sai và lặp lại dấu nhắc nhập liệu nhiều lần khi cần thiết để người dùng nhập dữ liệu đúng dạng thức và thuộc vùng giá trị yêu cầu.

Trong chương này, bạn đã tìm hiểu:

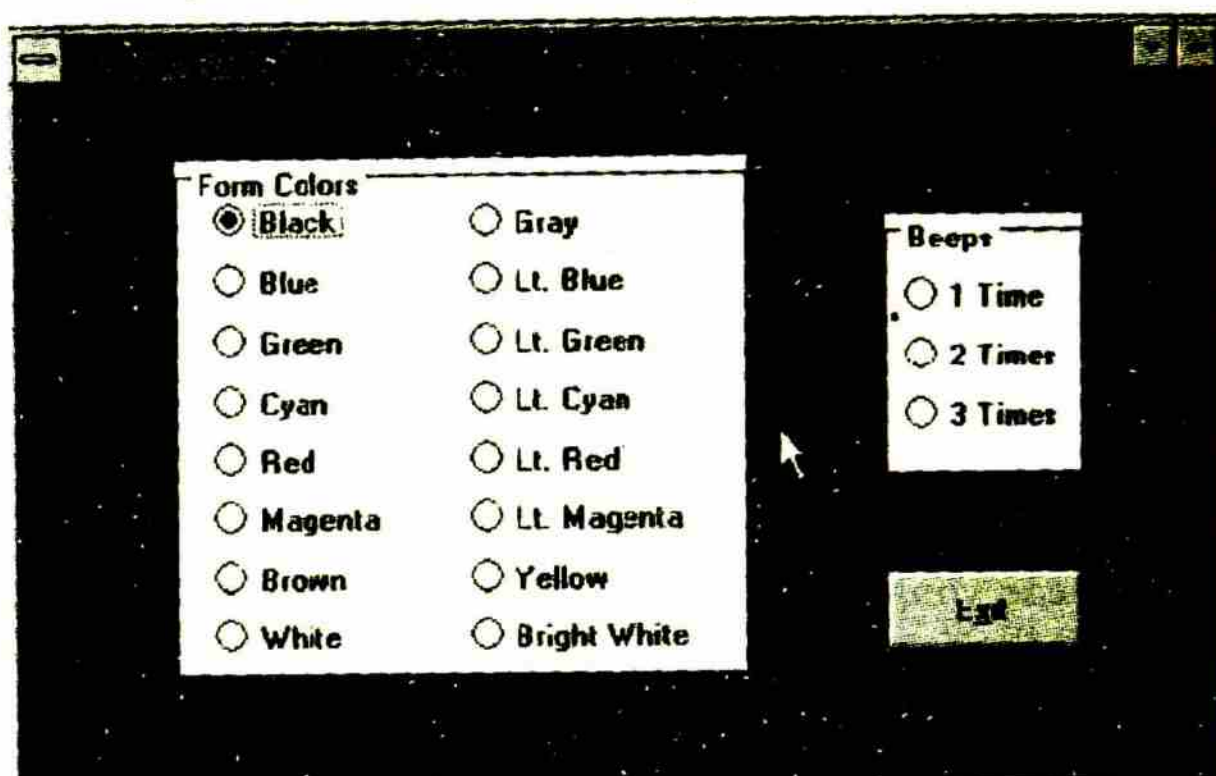
- Mảng giúp đơn giản quá trình định nghĩa nhiều biến
- Mảng điều khiển làm việc với nhiều điều khiển (thay vì nhiều thủ tục biến cố, bạn chỉ cần viết một thủ tục biến cố là đủ)
- Khi nào thì sử dụng ô chọn và khi nào sử dụng nút tùy chọn
- Nơi sử dụng và tạo nhóm nút tùy chọn để tập hợp các nút tùy chọn cho người dùng chọn.

Mô tả chương trình

Hình P6.1 mô tả chương trình PROJECT6.VBP. Bài thực hành này rất đơn giản, chỉ dùng hai nhóm nút tùy chọn được đóng khung và hai mảng điều khiển cho các nhóm nút tùy chọn. Những điều khiển của project gồm:

- Hai khung, một khung lưu 16 nút tùy chọn màu và khung còn lại lưu ba tùy chọn beep
- 16 nút tùy chọn trong khung bên trái dùng để chọn màu nền cho mẫu biểu

- Ba nút tùy chọn trong khung bên phải xác định số tiếng bíp người dùng muốn nghe
- Nút lệnh Exit kết thúc chương trình



Hình P6.1. *Màn hình ban đầu của chương trình.*

Chương trình sẽ đơn giản nhờ mảng điều khiển. Mảng điều khiển làm cho chương trình có nhiều điều khiển giống nhau thi hành các chức năng giống nhau. Không có mảng điều khiển, chương trình thực hành này sẽ trở nên rắc rối với ít nhất 20 thủ tục biến cố chỉ để thi hành cùng một nhiệm vụ mà hiện tại chỉ cần 3 thủ tục biến cố là đủ!

Hoạt động chương trình

Đầu tiên khi bạn chạy chương trình, màu nền mẫu biểu được ấn định màu đen bởi vì nút tùy chọn đầu tiên trong khung bên trái được chọn. Không thiết đặt tùy chọn tiếng bíp khi chương trình thi hành lần đầu tiên.

Bạn có thể chọn một trong các tùy chọn tiếng bíp cùng với các tùy chọn màu. Hãy nhớ rằng khi nút tùy chọn xuất hiện trong khung riêng biệt, các nút tùy chọn được nhóm để cho phép chọn từ mỗi nhóm. Không có khung, bạn chỉ có thể chọn một nút tùy chọn màu hoặc tiếng bíp, chứ không thể chọn cả hai.

Nhấp nút tùy chọn bíp thứ hai và bạn sẽ nghe hai tiếng bíp. Chọn một nút tùy chọn màu khác và màu nền mẫu biểu sẽ thay đổi theo.

Thủ tục biến cố

Ví dụ P6.1 trình bày các thủ tục biến cố của PROJECT6.VBP. Giá trị Index được gửi tới thủ tục biến cố optBeep_Click() và optColor_Click() và được dùng trong từng thủ tục để phát tiếng bíp hay thiết đặt màu mẫu biểu thích hợp.

Ghi chú

Hàm QBColor() là hàm có sẵn trong Visual Basic, nó trả lại mã màu thập lục phân để bạn không phải biết bất kỳ giá trị thập lục phân nào. Khi bạn sử dụng một giá trị số nguyên từ 0 đến 15 trong cặp ngoặc đơn QBColor(), Visual Basic ấn định màu phù hợp giá trị số đó. Các màu tương ứng với thứ tự hiện ra trên mẫu biểu. Giá trị Index cho 16 giá trị màu được đánh số từ 0 tới 15, cho nên thủ tục biến cố có thể sử dụng mã màu đó trong QBColor().

Ví dụ P6.1. *Quá trình thay đổi màu, phát tiếng bíp và thoát thủ tục biến cố.*

```
Sub optCol_Click (Index As Integer)
' One of the color option buttons
' triggered this event
frmOpt.BackColor = QBColor(Index)
End Sub
Sub optBeep_Click (Index As Integer)
' The beeping option buttons have Index
' properties that begin with 1 that
' correspond to the number of beeps.
Dim Ctr As Integer
Dim Delay As Long
For Ctr = 1 To Index
Beep
For Delay = 1 To 80000
' Do nothing but waste time...
Next Delay
Next Ctr
End Sub
Sub cmdExit_Click ()
End
End Sub
```

Mô tả

1: Các nút tùy chọn thay đổi màu tạo thành mảng điều khiển optCol, cho nên tên của thủ tục biến cố Click là optCol_Click(). Bất cứ khi nào người dùng nhấp 1 trong số 16 nút tùy chọn màu, thủ tục biến cố này sẽ thi hành.

2: Chú thích giúp giải thích mục đích thủ tục biến cố.

3: Chú thích giúp giải thích mục đích thủ tục biến cố.

4: Thiết đặt thuộc tính màu nền mẫu biểu với giá trị QBColor() của chỉ số Index tương ứng với nút tùy chọn đã nhấp.

5: Dừng thủ tục biến cố.

(5: Không có mảng điều khiển, mỗi nút tùy chọn sẽ đòi hỏi thủ tục biến cố riêng của nó.)

6: Nút tùy chọn bíp tạo mảng điều khiển tên optBeep, cho nên tên thủ tục biến cố Click là optBeep_Click(). Bất cứ khi nào người dùng nhấp một trong ba nút tùy chọn beep, thủ tục biến cố này sẽ thi hành.

7: Chú thích giúp giải thích mục đích thủ tục biến cố.

8: Chú thích giúp giải thích mục đích thủ tục biến cố.

9: Chú thích giúp giải thích mục đích thủ tục biến cố.

10: Định nghĩa một biến số nguyên được dùng cho biến điều khiển vòng lặp.

11: Định nghĩa một biến số nguyên dài được dùng để làm chậm lại vòng lặp.

12: Vòng lặp đủ thời gian làm cho phù hợp giá trị Index của nút tùy chọn, tùy thuộc cửa sổ Property có giá trị từ 1 đến 3.

13: Phát tiếng bíp ra loa máy tính.

14: Bắt đầu vòng lặp lớn làm chậm các tiếng bíp.

15: Chú thích chỉ là việc bên trong vòng lặp được lồng trong cùng.

(15: Vòng lặp lồng sẽ làm chậm tốc độ phát tiếng bíp của máy tính.)

16: Tiếp tục chiếm thời gian bằng cách lặp vòng lặp bên trong.

17: Phát lại tiếng bíp nếu tiếng bíp chưa hoàn thành.

18: Dừng thủ tục biến cố.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic.

Bạn có cảm thấy mệt mỏi với chương trình thay đổi tiếng bíp và màu không? Trình ứng dụng đơn giản mà bạn đang làm việc hướng đến môi trường và ngôn ngữ Visual Basic. Cho đến bây giờ bạn đã đọc một nửa tập sách. Bạn đã có đủ các công cụ lập trình trong Visual Basic để bắt đầu viết nhiều trình ứng dụng lớn hơn, và bạn sẽ làm điều đó trong chương tiếp theo.

Chương VII

Bài 13

Hàm xây dựng sẵn

- Tổng quan về hàm
- Hàm số
- Hàm chuỗi
- Hàm chung

Để xây dựng một căn nhà, có lẽ bạn sẽ mua các cửa sổ và những khung cửa ra vào đã làm sẵn thay vì tự tay đóng khung và làm chúng. Trừ khi bạn muốn dùng thời gian làm công việc đáng lẽ bạn không phải làm, sử dụng cửa sổ và khung cửa ra vào làm sẵn sẽ thúc đẩy nhanh quá trình xây dựng căn nhà và cho phép bạn tập trung khả năng của mình vào những việc khác (như phân chia không gian phòng ngủ để có chỗ đặt một máy tính, ...)

Tương tự, *hàm xây dựng sẵn* (built–in function) trong Visual Basic sẽ tiết kiệm thời gian viết mã lệnh trong thủ tục con để dùng chung. Hàm xây dựng sẵn làm việc giống một chương trình nhỏ thực hiện những công việc phổ biến để bạn có thể tập trung khả năng vào những công việc khác. Để làm tròn một giá trị chính xác đơn thành một giá trị nguyên integer, Visual Basic hỗ trợ không chỉ một mà tới ba hàm xây dựng sẵn.

Mách nước

Có nhiều hàm xây dựng sẵn sẽ trình bày trong tập sách này, chẳng hạn hàm Val(), hàm Chr$(). Thông thường, hàm kết thúc tên bằng dấu $ là hàm chuỗi, và hàm không có dấu $ trong tên của nó là hàm số. Bài này sẽ trình bày về cả hai loại hàm chuỗi và hàm số.

TỔNG QUAN VỀ HÀM

Khái niệm

Hàm chấp nhận một hay nhiều đối số và làm việc với các đối số này. Sau đó, hàm trả về một giá trị. Sử dụng hàm trong câu lệnh Visual Basic, có nghĩa là gọi hàm. Ví dụ, câu lệnh sau đây sẽ gọi hàm Val():

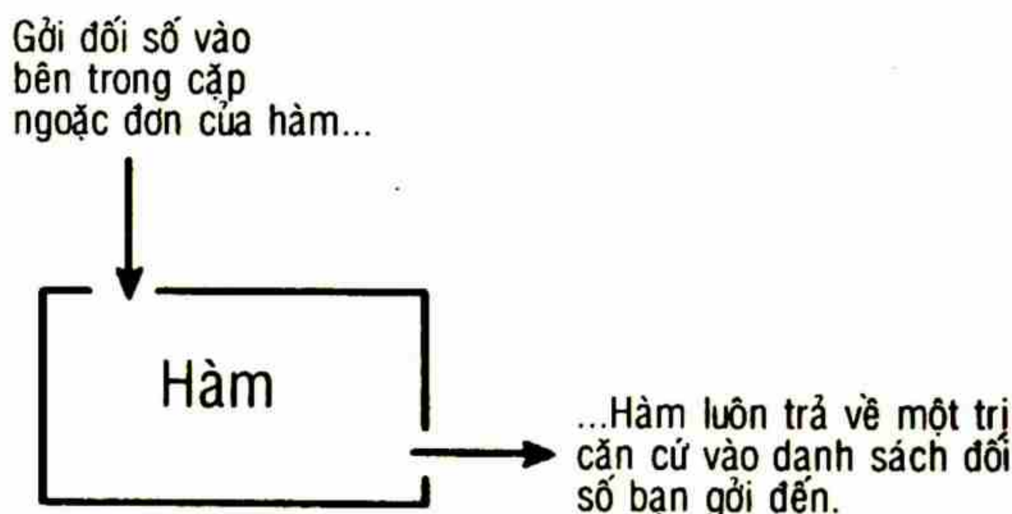
```
Num = Val(txtAmt.Text) ' Convert entry to number
```

Khái niệm mới

Đối số là một giá trị chuyển tới hàm.

Các dấu ngoặc đơn sau tên hàm giữ một hay nhiều đối số. Trong câu lệnh trước, txtAmt.Text là đối số. Một số hàm chẳng hạn như Val(), chỉ chấp nhận một đối số. Một số hàm nhận nhiều đối số. Có hàm lại không đòi hỏi bất kỳ đối số nào, và với những hàm này, cặp dấu ngoặc đơn có thể bỏ đi. Các đối số bên trong cặp dấu ngoặc đơn được gọi là danh sách đối số. Một hàm có thể không có đối số hoặc yêu cầu một hay nhiều đối số.

Xác định đối số cũng có nghĩa là chuyển đối số tới hàm. Hàm làm việc trên các giá trị đối số chuyển tới và đưa ra một số kiểu giá trị dựa trên những đối số đó. Hình 13.1 minh họa cách gởi vào hàm một hay nhiều đối số và sau khi sử dụng chúng, hàm trả lại một giá trị. Giá trị đó là kết quả của hàm, được gọi là giá trị trả về. Tóm lại, hàm nhận các đối số và trả về một giá trị.



Hình 13.1. *Hàm làm việc trên những đối số của nó và trả về một giá trị.*

Ghi chú

Một việc rõ ràng và rất có giá trị là một hàm luôn luôn trả về một giá trị và chỉ một giá trị mà thôi. Mặc dù danh sách đối số hàm đôi khi có thể chứa năm bảy đối số, hàm không bao giờ trả về nhiều hơn một giá trị.

Nếu hàm đòi hỏi nhiều đối số, hãy phân cách các đối số này bằng một hay nhiều dấu phẩy. Sau đây là hàm Right$() đòi hỏi hai đối số, MyName và 10, trong danh sách đối số của nó:

```
Last = Right$(MyName, 10) ' Strip off last name
```

Phải luôn thực hiện một điều gì đó với giá trị trả về của hàm. Sau đây là những việc có thể làm với giá trị trả về:

- Hiển thị giá trị trả về của hàm trên mẫu biểu bằng cách gán giá trị đó cho một điều khiển
- Gán giá trị trả về cho một biến
- Sử dụng giá trị trả về trong một biểu thức tính
- Sử dụng giá trị trả về như một đối số của hàm khác bằng cách lồng một hàm vào bên trong danh sách đối số của hàm khác

Nói chung, bất cứ điều gì có thể thực hiện với biến hay hằng thì cũng sẽ thực hiện được với giá trị trả về của hàm. Ví dụ, bạn đã học trong Bài 7 cách hàm Val() làm việc. Hàm Val() đổi đối số chuỗi (hay Variant) của nó thành một số. Các số được đổi là giá trị trả về của hàm Val(). Hàm Val() sẽ đổi đối số của nó thành giá trị trả về, vì vậy, không thể viết hàm Val() như sau:

```
Val(UserAddress) ' Invalid!
```

Hàm Val() chấp nhận chuỗi tên UserAddress và đổi chuỗi đó thành một số. Phải bảo đảm rằng chương trình sẽ thực hiện điều gì đó với số! Visual Basic cung cấp các hàm được định nghĩa trước, hay những thủ tục con xây dựng sẵn, nhận danh sách đối số và trả về giá trị để chương trình thực hiện điều gì đó với nó. Sau đây là bốn cách sử dụng hàm Val() khác nhau có sử dụng giá trị trả về :

```
lblHouseNum.Caption = Val(UserAddress) ' To a label
MyHouseNum = Val(UserAddress) ' To a variable
OldAge = 65 + Val(txtAge.Text) ' A calculation
Last = Right$(UserName, Val(Ans)) ' Another function
```

Đối số có thể là biểu thức. Hàm Val() dưới đây, đầu tiên sẽ nối hai biểu thức chuỗi được sử dụng như đối số, và sau đó trả về giá trị số được chuyển đổi từ đối số của chuỗi đó:

```
Number = Val(aStr1 & aStr2)
```

Ghi chú

Kiểu dữ liệu của danh sách đối số không phải lúc nào cũng phù hợp với kiểu dữ liệu của giá trị trả về của hàm. Ví dụ, một hàm có thể nhận đối số chuỗi và trả về một số. Một hàm có thể đòi hỏi cả đối số chuỗi lẫn số, và trả về một chuỗi.

Hàm xây dựng sẵn không bao giờ thay đổi đối số và sử dụng đối số của chúng để tạo ra một giá trị mới – Đó là giá trị trả về.

Ôn lại

Có nhiều hàm xây dựng sẵn trong Visual Basic mà chương trình có thể gọi. Chương trình gọi hàm phải thực hiện điều gì đó với giá trị trả lại từ hàm. Một hàm có thể đòi hỏi một hay nhiều đối số. Nếu gửi danh sách đối số tới hàm, hàm sẽ xử lý những đối số này và trả về một giá trị.

HÀM SỐ

Khái niệm

Visual Basic có nhiều hàm số giúp tiết kiệm thời gian lập trình khi làm việc với số. Đó là các hàm chuyển đổi, hàm khoa học và hàm lượng giác.

Đừng lo lắng về việc phải nhớ từng hàm vào lúc này. Hãy cố để có một khái niệm tổng quát về các loại hàm trong Visual Basic. Hàm giúp bạn tiết kiệm thời gian. Nếu cần đổi một số hay tính toán công thức, đây chính là cơ hội để sử dụng hàm trong Visual Basic. Ví dụ, không có lý do gì để viết các hàm lượng giác cao cấp như hàm sin, bởi vì Microsoft đã viết sẵn một hàm như vậy.

Ghi chú

Thực tế không phải ai cũng thích toán học để sử dụng và đánh giá đúng lợi ích các hàm số trong Visual Basic. Nhưng lợi ích rõ ràng nhất là bạn không bao giờ phải viết mã lệnh thực hiện những công việc tương tự.

Chuyển đổi số

Visual Basic hỗ trợ ba hàm để đổi các số chính xác đơn và chính xác kép thành những giá trị nguyên. Khi viết trình ứng dụng, có lẽ cần làm tròn các số giảm hay tăng thêm một đơn vị. Bảng 13.1 trình bày ba hàm đổi số nguyên.

Bảng 13.1. *Hàm đổi số nguyên*

Hàm	Mô tả
CInt()	Làm tròn giá trị thập phân lớn hơn hoặc bằng 5 thành số nguyên cao nhất kế tiếp.
Fix()	Cắt bỏ phần thập phân.
Int()	Làm tròn số thành số nguyên nhỏ hơn hoặc bằng đối số của nó.

Cả hai hàm Fix() và Int() đều đề cập đến số nguyên dương. Hai lệnh sau đều trả về và lưu kết quả là 14:

```
ans1 = Int(14.6)
ans2 = Fix(14.6)
```

Hàm Fix() và Int() hoạt động khác nhau đối với số âm. Những ghi chú bên phải hai lệnh sau chỉ ra giá trị trả về của hàm:

```
ans3 = Int(-14.6) ' Stores -15
ans4 = Fix(-14.6) ' Stores -14
```

Số âm nhỏ hơn hay bằng –14.6 là –15; giá trị trả về cho hàm Int() hiện ra trong phần ghi chú của câu lệnh đầu tiên.

Hàm Clnt() trả về các số thực sự như sau :

```
ans5 = CInt(14.1) ' Stores 14
ans6 = CInt(14.6) ' Stores 15
ans7 = CInt(-14.1) ' Stores -14
ans8 = CInt(-14.8) ' Stores -15
```

Có thể sử dụng các hàm tương tự Clnt() để đổi những đối số thuộc kiểu dữ liệu bất kỳ thành một kiểu dữ liệu bất kỳ khác. Bảng 13.2 cho biết những hàm đổi kiểu dữ liệu còn lại.

Bảng 13.2. *Các hàm chuyển đổi kiểu dữ liệu*

Hàm	Diễn giải
Ccur()	Đổi đối số thành kiểu dữ liệu currency tương ứng.
CDbl()	Đổi đối số thành kiểu dữ liệu chính xác kép tương ứng.
CLng()	Đổi đối số thành kiểu dữ liệu số nguyên dài tương ứng.
CSng()	Đổi đối số thành kiểu dữ liệu chính xác đơn tương ứng.
CStr()	Đổi đối số thành kiểu dữ liệu chuỗi tương ứng.
CVar()	Đổi đối số thành kiểu dữ liệu Variant tương ứng.

Lưu ý

Nếu đối số được đổi không phù hợp với kiểu dữ liệu đích, Visual Basic đưa ra thông báo lỗi Overflow. Muốn chắc chắn nên dùng một lệnh If trước đó để kiểm tra đối số có nằm trong các vùng thuộc kiểu dữ liệu được đổi hay không.

Thông thường, lệnh gán sau sẽ lưu .1428571 trong nhãn lblValue:

```
lblValue.Caption = (1 / 7) ' Assigns .1428571
```

Tuy nhiên, lệnh sau sẽ thêm phần thập phân vào kết quả với các phép tính chính xác hơn:

```
lblValue.Caption = CDbl(1 / 7) ' Assigns .142857142857143
```

Đừng quan tâm những hệ đếm số khác! Visual Basic còn hỗ trợ bốn hàm đổi đối số ở hệ đếm thập phân thành bát phân và thập lục phân. Hãy sử dụng những hàm này nếu viết các ứng dụng nâng cao liên quan đến những hệ đếm số khác.

Hai hàm thập lục phân, Hex() và Hex$(), đổi đối số kiểu số thành giá trị thập lục phân Variant hay chuỗi tương ứng. Tương tự, hai hàm bát phân, Oct() và Oct$(), cũng đổi đối số kiểu số thành giá trị bát phân Variant hoặc chuỗi tương ứng.

Nếu nghĩ rằng bạn không bao giờ sử dụng các hàm này, hãy quên chúng đi! Những hàm này, đặc biệt hàm thập lục phân, thường được các lập trình viên dùng để viết chương trình ở mức hệ thống như chương trình bảo trì đĩa và bộ nhớ.

Hàm Asc() đổi đối số chuỗi thành giá trị trong bảng ASCII thích hợp. Nhìn chung, bạn sẽ chuyển một chuỗi ký tự (hay một giá trị

Variant có thể tạo thành một chuỗi) với hàm Asc(). Nếu đối số có nhiều ký tự, hàm Asc() sẽ bỏ qua tất cả ký tự sau ký tự đầu tiên.

Lệnh sau sẽ lưu giá trị 65 trong biến Initial bởi vì 65 là ký tự *A* trong bảng ASCII :

```
Initial = Asc("A")
```

Bảng 13.3 sẽ liệt kê các hàm toán học còn lại mà Visual Basic hỗ trợ. Có nhiều hàm khoa học và toán học năm thì mười họa mới được sử dụng trừ khi bạn viết trình ứng dụng chuyên về toán.

Bảng 13.3. *Các hàm khoa học và toán học*

Hàm	Diễn giải
Abs()	Trả về giá trị tuyệt đối của đối số.
Atn()	Trả về góc cung của đối số theo radian.
Cos()	Trả về giá trị cos của đối số theo radian.
Exp()	Trả về cơ số logarit tự nhiên của đối số.
Log()	Trả về logarit tự nhiên của đối số.
Sin()	Trả về giá trị sin của đối số theo radian.
Sqr()	Trả về căn bậc hai của đối số.
Tan()	Trả về tang của đối số theo radian.

Thuật ngữ mới

Pi là một giá trị xấp xỉ 3.14159 và được sử dụng nhiều trong các phép tính diện tích.

Lời nhắc

Nếu bạn tính giá trị lượng giác trên đối số theo độ thay vì radian, hãy nhân đối số với pi và chia cho 180. Biểu thức sau sẽ gán giá trị sin của 38 độ cho biến sVal:

```
sVal = Sin(38 * 3.14159 / 180)
```

Ôn lại

Các hàm số sẽ giúp bạn viết chương trình khi cần các phép tính toán học trong mã lệnh. Nhiều trình ứng dụng trong kinh doanh, đồ họa và nghiên cứu đôi khi sử dụng những thủ tục phức tạp, và các hàm toán học Visual Basic sẽ giúp rút ngắn thời gian lập trình đồng thời tăng tính chính xác cho mã lệnh.

HÀM CHUỖI

Khái niệm

Visual Basic có nhiều hàm chuỗi cho phép tổ hợp dữ liệu chuỗi tốt hơn so với nhiều ngôn ngữ lập trình khác. Hàm chuỗi cho phép bạn cắt một số ký tự từ một chuỗi dữ liệu và thay đổi chuỗi thành dạng khác.

Ngôn ngữ lập trình Visual Basic cung cấp hai loại chuỗi là chuỗi có chiều dài cố định và chuỗi có chiều dài phụ thuộc vào biến đã định nghĩa. Xử lý chuỗi trong Visual Basic dễ hơn trong các ngôn ngữ lập trình khác hiện nay.

Bạn đã thấy hàm Chr$() đổi số thành ký tự ASCII tương ứng của nó. (Ngoài ra còn có hàm Chr() trả về ký tự theo kiểu dữ liệu Variant.) Hàm Str$() (và hàm Str() trả về giá trị theo kiểu dữ liệu Variant) dùng để đổi số thành chuỗi.

Mách nước

Hãy đổi số thành chuỗi khi muốn hiển thị giá trị số trong hộp thông báo.

Giả sử tuổi người dùng được lưu trong biến số tên Age. Nếu muốn hiển thị tuổi của người dùng trong hộp thông báo, bạn không thể thực hiện mà không có bước chuyển đổi giá trị tuổi thành chuỗi như sau:

```
MsgBox "Your age is now" & Str$(Age)
```

Các hàm đổi chuỗi sẽ đổi đối số chuỗi thành chuỗi ký tự hoa hay thường. Bảng 13.4 sẽ liệt kê các hàm này.

Bảng 13.4. *Các hàm đổi chuỗi*

Hàm	Diễn giải
LCase()	Trả về đối số chuỗi như một kiểu dữ liệu Variant mà tất cả ký tự là chữ thường.
LCase$()	Trả về đối số chuỗi như một kiểu dữ liệu chuỗi mà tất cả ký tự là chữ thường.
UCase()	Trả về đối số chuỗi như một kiểu dữ liệu variant mà tất cả ký tự là chữ hoa.
UCase$()	Trả về đối số chuỗi như một kiểu dữ liệu chuỗi mà tất cả ký tự là chữ hoa.

Nếu đối số đã có một hay nhiều chữ hoa, các hàm đổi chuỗi thành chuỗi ký tự hoa sẽ không đổi những ký tự này. Phần chú thích trong các câu lệnh sau cho biết giá trị trả về của hàm:

```
lowerName = LCase("Larry") ' Returns larry
upperName = UCase("Smythe") ' Returns SMYTHE
```

Khái niệm mới

Chuỗi con là một phần của chuỗi.

Visual Basic có ba hàm chuỗi trả về các chuỗi con bên trái, ở giữa, hay bên phải của một chuỗi. Đó là các hàm Left$(), Mid$(), và Right$(). Tất cả hàm này được dùng để trả lại giá trị chuỗi con. Sau đây là dạng thức những hàm này:

```
Left$(StringVal, length)
Right$(StringVal, length)
Mid$(StringVal, startVal, length)
```

Ghi chú

Hàm Mid$() được gọi là hàm midstring.

Hàm Left$() trả về số ký tự *length* trong chuỗi *StringVal*. *StringVal* phải là một hằng hay biến chuỗi. Cũng như thế, hàm Right$() trả về *length* ký tự trong chuỗi *StringVal*. Hàm Mid$() trả về *length* ký tự trong chuỗi *StringVal* với ký tự bắt đầu tại vị trí *startVal*.

Ghi chú trong các câu lệnh sau đây sẽ cho biết giá trị trả về của hàm:

```
Title = "Something Good!"
lTitle = Left$(Title, 4) ' Some
rTitle = Right$(Title, 5) ' Good!
mTitle = Mid$(Title, 5, 8) ' thing Go
```

Hình 13.2 sẽ minh họa cách Visual Basic tìm các chuỗi con trong những giá trị chuỗi dài hơn.

```
Something Good!
LeftS(Title, 4)

Something Good!
Right$(Title, 5)

Something Good!
Mid$(Title, 5, 8)
```

Hình 13.2. *Các hàm lấy chuỗi con từ chuỗi.*

Hàm Len() trả về số ký tự được lưu trong các chuỗi. Thông thường phải hiệu chỉnh kích cỡ điều khiển để giữ các chuỗi dài. Có thể sử dụng hàm Len() xác định độ rộng cho một điều khiển. Lệnh sau lưu giá trị 14 vào biến tên Length:

```
length = Len("Blue and Green")
```

Ghi chú

Hàm Len() làm việc khá ăn ý với các giá trị số. Nếu bạn chuyển một biến số, hằng hay biểu thức vào hàm Len(), hàm Len() sẽ trả về tổng số vùng nhớ cần để giữ giá trị số đó.

Hàm LTrim$() và hàm RTrim$() giúp cắt các khoảng trống ở đầu hoặc cuối một chuỗi. Nếu các khoảng trống không tồn tại, hàm LTrim$() và RTrim$() sẽ không làm gì cả. Hàm LTrim$() trả về chuỗi đối số không có bất kỳ khoảng trống ở đầu. Hàm RTrim$() trả về chuỗi đối số không có bất kỳ các khoảng trống ở cuối. Hàm Trim$() xử lý cả hai công việc bằng cách cắt các khoảng trống ở đầu và cuối một chuỗi.

Sau đây là dạng thức các hàm cắt chuỗi:

```
LTrim[$](StringExpression)
RTrim[$](StringExpression)
Trim[$](StringExpression)
```

Cũng có hàm Variant tương đương được gọi là LTrim(), RTrim(), và Trim().

Các lệnh sau sẽ cắt những khoảng trống ở đầu, cuối hoặc cả hai phía trong các chuỗi:

```
LStr1 = LTrim$(" Jonah") ' Stores Jonah
RStr2 = RTrim$("Jonah ") ' Stores Jonah
AnySt3 = Trim$(" Jonah ") ' Stores Jonah
```

Không có hàm cắt, các khoảng trống được sao chép vào những biến đích cùng với tên Jonah.

Phần trước, bạn đã học cách sử dụng hàm Str$() để chuyển một số thành chuỗi. Bởi vì hàm Str$() luôn luôn đổi số dương thành chuỗi với một khoảng trống ở đầu (nơi dấu cộng tưởng tượng sẽ xuất hiện), có thể kết hợp hàm LTrim$() với Str$() để loại bỏ khoảng trống đó. Lệnh đầu tiên trong hai lệnh sau đây sẽ lưu các khoảng trống vào trong st1. Lệnh thứ hai sử dụng LTrim$() để loại khoảng trống trước khi lưu chuỗi vào st2.

```
st1 = Str$(234) ' Stores " 234"
st2 = LTrim$(Str$(234)) ' Stores "234"
```

Ôn lại

Hàm chuỗi chuyên xử lý, tổ hợp các chuỗi và trả về giá trị dựa trên đối số chuỗi. Khi viết nhiều chương trình lớn, bạn cần trích các chuỗi con và chuyển chuỗi thành dạng khác để hiển thị những giá trị chuỗi trên mẫu biểu và trong các điều khiển.

HÀM CHUNG

Khái niệm

Có một số hàm chung không nằm trong nhóm hàm số hay hàm chuỗi. Những hàm chung này tạo thêm sức mạnh về mặt kỹ xảo lập trình của bạn. Phần này sẽ giải thích cách dò các kiểu dữ liệu bằng cách sử dụng các hàm Is...() và VarType(). Có lẽ bạn chưa thấy lợi ích của những hàm này, nhưng bản thân bạn đã quen thuộc với chúng. Trong Chương 8, bạn sẽ học cách trả lại dữ liệu hay nhận dữ liệu giữa các thủ tục con trong Visual Basic, và đôi khi bạn cần biết kiểu dữ liệu nào đã chuyển cho bạn. Bạn cần sử dụng một trong những hàm được chỉ ra ở đây.

Bảng 13.5 trình bày các hàm xác định kiểu dữ liệu, thường được gọi là hàm Is...(). Khi bạn lưu giá trị trong biến Variant (có thể chấp nhận kiểu dữ liệu bất kỳ), các hàm trong Bảng 13.5 trả về kết quả True hay False, cho biết liệu đối số nào có thể được chuyển thành kiểu dữ liệu dự định.

Bảng 13.5. *Các hàm xác định kiểu dữ liệu Is...()*

Tên hàm	Diễn giải
IsDate()	Xác định đối số của nó có thể được chuyển thành một ngày hợp lệ.
IsEmpty()	Xác định đối số của nó đã được khởi tạo với dữ liệu bất kỳ kể từ định nghĩa ban đầu của đối số.
IsNull()	Xác định đối số của nó giữ một giá trị rỗng Null (dưới dạng chuỗi trống).
IsNumeric()	Xác định đối số của nó giữ một giá trị có thể đổi thành một số hợp lệ.

Mã lệnh sau đảm bảo rằng chuỗi hợp lệ sẽ xuất hiện trong biến tên ans trước khi áp dụng hàm Ucase$():

```
If IsString(ans) Then
NewAns = UCase$(ans)
Else
 MsgBox "You didn't type a letter!"
End If
```

Biến Empty khác với các giá trị Null và 0. Biến Empty chỉ ra rằng không có gì được nhập vào biến. Hàm IsNull() kiểm tra liệu đối số điều khiển của nó có giá trị rỗng (Null).

Mách nước

Sử dụng hàm IsNull() để xem liệu một điều khiển hay một trường trên mẫu biểu có chứa dữ liệu. Sử dụng hàm IsEmpty() thích hợp với các biến.

Hàm VarType() xác định kiểu dữ liệu đối số của nó. Bảng 13.6 liệt kê các giá trị trả về từ hàm VarType(). VarType() trả về một trong 9 giá trị được liệt kê trong bảng.

Bảng 13.6. *Các giá trị trả về của hàm VarType()*

Giá trị trả về	Diễn giải
0	Chỉ đối số là Empty.
1	Chỉ đối số là Null.
2	Chỉ đối số là Integer.
3	Chỉ đối số là Long.
4	Chỉ đối số là Single.
5	Chỉ đối số là Double.
6	Chỉ đối số là Currency.
7	Chỉ đối số là Date.
8	Chỉ đối số là String.

Tóm tắt

Ví dụ 13.1 đảm bảo rằng người dùng đã nhập giá trị vào điều khiển hộp nhập txtAge.

Ôn lại

Các hàm chung trả về giá trị cho biết nội dung và kiểu dữ liệu. Bạn có thể sử dụng những hàm này để xem liệu các biến và những điều khiển đã được khởi tạo trước khi tính toán với giá trị đó không.

Ví dụ 13.1. *Kiểm tra để chắc chắn rằng người dùng đã nhập vào một giá trị.*

```
1: If IsNull(txtAge.Text) Then
2: MsgBox "You didn't type anything!"
3: Else
4: MsgBox "Thanks for entering a value."
5: End If
```

Phân tích

Dòng 1 bảo đảm rằng người dùng đã nhập một cái gì đó trong điều khiển có tên txtAge. Mã lệnh kế tiếp có thể sử dụng hàm VarType() để bảo đảm rằng người dùng đã nhập vào giá trị có kiểu dữ liệu đúng.

Bài tập

Kiến thức tổng quát

1. Đối số là gì?
2. Đúng hay Sai: Giá trị trả về của hàm phải phù hợp với kiểu dữ liệu của danh sách đối số.
3. Đúng hay Sai: Một số hàm không đòi hỏi đối số, một số đòi hỏi một hay nhiều đối số.
4. Đúng hay Sai: Các đối số của hàm phải phù hợp với tất cả các kiểu dữ liệu.
5. Tại sao có tới ba hàm nguyên?
6. Hãy cho biết giá trị Visual Basic lưu trong biến Ans trong những câu lệnh gán sau đây?
   A. Ans = Int(61.32)
   B. Ans = Int(–61.32)
   C. Ans = Fix(–61.32)
   D. Ans = Cint(421.51)
7. Đúng hay Sai: Int(), CInt(), và Fix() đều trả về cùng giá trị với các đối số dương.
8. Hàm nào thi hành ngược với hàm Chr$()?
9. Giá trị nào sẽ xuất hiện trong biến chuỗi có tên là AList sau những phép gán sau?
   A. AList = UCase$("AbCdEfG")
   B. AList = LCase$("AbCdEfG")
10. Lệnh gán sau đây lưu điều gì trong biến Variant có tên là Anything?
    Anything = Str$(Val("78.1"))
11. Hãy viết các giá trị được lưu trong từng câu lệnh gán sau đây:
    A. AStr = Left$("Sams", 1)
    B. AStr = Left$("Sams", 3)
    C. AStr = Right$("Sams", 1)
    D. AStr = Right$("Sams", 3)
    E. AStr = Mid$("Sams", 2, 3)

12. Đúng hay Sai: Phụ thuộc vào các đối số, hàm Mid$() có thể trả về cùng giá trị như hàm Left$() và Right$().

13. Đúng hay Sai: Cả hai kết quả sau đều cùng giá trị khi dùng trong một biểu thức:

```
Chr$(67)
Asc("C")
```

14. Tên của hàm chuyển số thành chuỗi là gì?

Viết mã lệnh...

15. Giả sử bạn cần hiển thị một số giá trị của biến bên trong chuỗi dấu nhắc của hộp nhận dữ liệu (input box). Hãy mô tả cách bạn nối một biến như thế tới chuỗi dấu nhắc?

16. Cách bạn có thể tìm ra tổng số vùng nhớ bị chiếm bởi 250 biến chính xác kép?

17. Hãy viết mã lệnh sử dụng hộp nhận dữ liệu để nhận một số từ người dùng và sau đó sử dụng hộp thông báo để hiển thị số.

18. Hãy viết mã lệnh sử dụng hai hộp nhận dữ liệu để hỏi người dùng về họ và tên, sau đó hiển thị trong hộp thông báo tổng số ký tự của cả họ và tên.

19. Viết một câu lệnh gán lưu giá trị ASCII của "P" trong một biến có tên là ValAsc.

Tìm lỗi kỹ thuật

20. An Huy nhận một thông báo tràn số khi đang thử lệnh gán sau trong một biến integer có tên là Weight:

```
Weight = CInt(32768 * Num)
```

Hãy mô tả vấn đề của An Huy.

Phần nâng cao

Thêm vào Ví dụ 13.1 để bảo đảm rằng người dùng đã nhập vào một số integer trong điều khiển txtAge.Text.

Hãy viết chương trình lưu 256 kí tự ASCII (từ ASCII 0 tới ASCII 255) vào một mảng chuỗi được định nghĩa là giữ 256 phần tử.

Bài 14

Các hàm ngày, giờ và dạng thức

- Lấy ngày giờ hệ thống
- Đặt ngày giờ trong Visual Basic
- Xác định thời gian trôi qua
- Giá trị ngày và thời gian nối tiếp
- Định dạng với hàm Format()

Bài này trình bày các hàm ngày giờ và dạng thức xây dựng sẵn trong Visual Basic. Đây là bài đầu tiên bạn làm việc trực tiếp với kiểu dữ liệu Variant mặc dù bạn đã từng gặp nhiều dữ liệu kiểu Variant trong các chương trước.

Ngoài ra, bài này còn thảo luận nhiều lệnh ngày giờ giúp bạn làm việc với các giá trị ngày giờ. Giá trị ngày giờ luôn biến động, khó ghi lại trong các giao dịch và báo cáo, cho nên bạn phải có khả năng viết các chương trình tìm và làm việc với những giá trị như thế. Visual Basic hỗ trợ một trong những thư viện toàn diện nhất về hàm ngày giờ.

Ghi chú

Hiểu cách Visual Basic lưu các giá trị ngày và giờ trong bộ nhớ là phần quan trọng nhất.

Khả năng định dạng của Visual Basic thật là tuyệt vời. Bạn sẽ thấy các bảng định dạng giúp bạn trình bày chương trình dưới dạng mong muốn. Visual Basic hỗ trợ khả năng định dạng ngày, giờ, chuỗi và số.

LẤY NGÀY GIỜ HỆ THỐNG

Khái niệm

Các hàm Now(), Date$() và Time$() (và những hàm bà con của chúng, là Date(), Time()) căn cứ vào đồng hồ và lịch trong hệ thống máy tính để lấy ngày giờ hiện hành cho chương trình bạn đang sử dụng.

Cách đây khoảng 8 năm, máy tính không thể nhớ ngày giờ sau khi tắt máy. Đó là điều xảy ra trong một thời gian dài! Một số người đã đưa ra ý kiến tuyệt vời là đặt một pin máy tính để máy tính nhớ ngày giờ mỗi lần bật máy. Dĩ nhiên, pin đã được sử dụng trong đồng hồ xem giờ nhiều năm trước đó, nhưng các nhà chế tạo máy tính quá bận rộn cho nỗ lực lưu trữ 360K không gian bộ nhớ vào đĩa mềm 5 ¼–inch, nên họ chưa thể thực hiện công việc đặt pin vào máy tính lúc đó.

Visual Basic giúp chương trình Windows truy xuất các giá trị ngày giờ được lưu bên trong máy tính. Giả sử ngày giờ đã xác định đúng, bạn có thể gán các giá trị ngày giờ hệ thống vào các biến để hiển thị chúng trong điều khiển.

Khái niệm mới

24–hour time ***là cách tính giờ từ 0 đến 24.***

Một số giá trị thời gian trả lại từ các hàm ngày giờ hiển thị theo dạng 24 giờ. Giá trị thời gian 12 giờ hiển thị cùng với a.m. hay p.m, có thể đổi thành thời gian 24 giờ bằng cách cộng 12 với tất cả giá trị giờ sau 12:59 pm. Nghĩa là 3:45 am là 3:45, nhưng 3:45 pm là 15:45.

Ghi chú

Có thể sử dụng hàm Format() để thay đổi thời gian từ dạng 24 giờ thành 12 giờ bằng cách sử dụng thêm các từ am và pm.

Các hàm Now(), Date$(), Date(), Time$(), và Time() là những hàm không chấp nhận đối số. Như vậy, Visual Basic sẽ bỏ các dấu ngoặc đơn nếu chúng được nhập sau những hàm này. Tuy nhiên, sách vẫn sử dụng các dấu ngoặc đơn sau hàm để nhắc nhở rằng chúng là các hàm xây dựng sẵn.

Hàm Now() trả về cả hai giá trị ngày và giờ với kiểu dữ liệu Variant theo dạng sau đây:

mm/dd/yy hh:mm:ss [AM][PM]

Trong đó, *mm* là 2 số biểu thị tháng, *dd* là 2 số biểu thị ngày, và *yy* là 2 số biểu thị năm. Trong giá trị trả về của hàm Now(), giờ sẽ xuất hiện ở bên phải ngày và sử dụng đồng hồ 12 giờ thay vì 24 giờ như trong hàm Time$() và Time(). *hh* là 2 số biểu thị giờ, *mm* là 2 số biểu thị phút, và *ss* là 2 số biểu thị giây. Visual Basic sẽ lưu giá trị ngày giờ này bên trong bộ nhớ như một giá trị chính xác kép bởi vì chỉ giá trị chính xác kép mới đủ sức để giữ thông tin nhiều như thế. Mặc dù Visual Basic lưu ngày giờ dưới dạng giá trị chính xác kép, hàm Now() vẫn có thể trả về giá trị ngày giờ theo dạng đã định với kiểu dữ liệu Variant.

Nếu xác lập International trong Windows đòi hỏi cách định dạng ngày giờ khác, những hàm này sẽ trả về giá trị ngày giờ phù hợp với xác lập đó.

Hàm Date$() trả về chuỗi ngày hệ thống theo dạng sau:

mm–dd–yyyy

Date() trả về giá trị theo kiểu Variant. Điểm khác biệt giữa Date() và Date$() là hàm Date() không trả về số 0 ở đầu khi các giá trị ngày, tháng nhỏ hơn 10. Hàm Date() không thêm 20 vào năm, và hàm Date() chèn dấu / thay vì dấu – giữa các giá trị ngày.

Mách nước

Để chặt chẽ với năm 2000, hãy sử dụng hàm Date$() trả về một năm đầy đủ 4 ký số khi sử dụng chương trình trong thế kỷ mới.

Hàm Time$() trả về thời gian hệ thống theo kiểu chuỗi ở dạng 24 giờ:

hh:mm:ss

Trong đó *hh* là giờ (từ 00 tới 23), *mm* là phút (từ 00 tới 59), và *ss* là giây (từ 00 tới 59).

Hàm Time() trả về giờ hệ thống theo kiểu Variant ở dạng 12 giờ :

hh:mm:ss [AM] [PM]

Trong đó, ký hiệu AM hoặc PM sau giờ hệ thống sẽ cho biết thời gian chính xác trong ngày.

Tóm tắt

Ví dụ 14.1 có nhiều câu lệnh gán mà ghi chú của nó sẽ mô tả giá trị ngày giờ được hàm trả lại.

Ôn lại

Các hàm ngày giờ trả về giá trị thời gian hệ thống của máy tính để sử dụng và hiển thị chúng trong chương trình. Những hàm như Now(), Date(), hay Time() (hoặc Date$() và Time$()) sẽ trả về các dạng ngày giờ khác nhau.

Ví dụ 14.1. *Truy xuất các giá trị ngày giờ.*

```
1: lblNow.Caption = Now() ' 7/9/97 07:48 PM
2: lblDate1.Caption = Date() ' 7/9/97
3: lblDate2.Caption = Date$() ' 07/09/1997
4: lblTime1.Caption = Time() ' 07:48:24 PM
5: lblTime2.Caption = Time$() ' 19:48:24
```

Phân tích

Các lệnh gán giả sử rằng ngày và giờ hiện hành là July 9, 1997 lúc 7:48:24 tối. Các dạng ngày giờ khác nhau để bạn lựa chọn cách hiển thị giá trị ngày giờ trong nhãn.

ĐẶT NGÀY GIỜ TRONG Visual Basic

Khái niệm

Các lệnh Date và Time (không phải hàm) cho phép thay đổi giá trị ngày giờ của máy tính trong một trình ứng dụng Visual Basic. Giá trị ngày giờ vẫn duy trì cho đến khi bạn thay đổi lại chúng bằng các lệnh Date, Time của Visual Basic hoặc bằng các lệnh của DOS hay Windows.

Lệnh DOS DATE, TIME cho phép kiểm tra và thiết đặt ngày giờ trong máy tính. Chương trình Control Panel trong Windows bên trong nhóm Settings cho phép đặt ngày giờ trong Windows. Hãy kiểm tra ngày giờ trong máy tính hàng tháng và như vậy sẽ bảo đảm tính chính xác của chúng.

Visual Basic có các lệnh Date và Time đặt giá trị ngày giờ của đồng hồ và lịch để bàn trong máy tính. Hãy bỏ các dấu ngoặc đơn trong lệnh Date và Time, nếu không Visual Basic sẽ nghĩ rằng bạn đã sử dụng không đúng các hàm tương ứng.

Sau đây là các dạng thức của lệnh Date và Time:

```
Date[$] = dateExpression
```

Và

```
Time[$] = timeExpression
```

Như với các hàm tương ứng, có hai phiên bản cho từng lệnh, và dấu $ sẽ phân biệt hai phiên bản. Nếu không dùng dấu $ ở cuối, bạn phải nhập vào *dateExpression* một giá trị ngày thuộc kiểu chuỗi hay kiểu ngày. Tất cả các lệnh sau đều đặt ngày hiện hành của máy tính thành July 9, 1997:

```
Date = 7/9/1997
Date = 07/9/97
Date = July 9, 1997
Date = Jul 9, 1997
Date = 9-Jul-1997
Date = 9 July 1997
Date = 9 July 97
```

dateExpression phải là một ngày hợp lệ từ January 1, 1980 đến December 31, 2099, bằng không Visual Basic sẽ phát sinh lỗi.

Nếu xác định dấu $ ở cuối (như Date$), bạn có thể chỉ nhập *dateExpression* theo các dạng sau :

```
Date$ = 07-9-97
Date$ = 7-9-1998
Date$ = 7/9/97
Date$ = 7/9/1997
```

Do sở hữu nhiều dạng ngày, Visual Basic dễ chấp nhận dạng ngày bất kỳ mà bạn đã nhập.

Nếu bạn không xác định dấu $ ở cuối, bạn có thể nhập vào *timeExpression* hoặc theo đồng hồ 12 giờ hoặc theo đồng hồ 24 giờ với các dấu nháy kép như sau:

```
Time = "7:48 PM"
```

Hay

```
Time = "19:48"
```

Nếu bạn xác định dấu $ ở cuối (như Time$), có thể nhập vào *timeExpression* theo các dạng bất kỳ này:

```
hh
hh:mm
hh:mm:ss
```

Bạn phải sử dụng một giá trị đồng hồ 24 giờ khi dùng *Time$*.

Tóm tắt

Ví dụ 14.2 trình bày mã lệnh cho phép người dùng thay đổi ngày giờ.

Ôn lại

Các lệnh Date, Date$, Time và Time$ cho phép bạn thay đổi xác lập ngày giờ hệ thống. Những xác lập này sẽ duy trì ảnh hưởng cho đến khi bạn hay người dùng thay đổi chúng.

Ví dụ 14.2. *Hãy để người dùng thay đổi ngày giờ.*

```
Dim newDate As Variant
Dim newTime As Variant
MsgBox "The current date is " & Date$ ' Calls function
newDate = InputBox("What do you want to set the date to?")
If IsDate(newDate) Then
Date$ = newDate
End If ' Don't do anything if a good date isn't entered
MsgBox "The date is now " & Date$
MsgBox "The current time is " & Time$ ' Calls function
newTime = InputBox("What do you want to set the time to?")
If IsDate(newTime) Then
Time$ = newTime
End If ' Don't do anything if a good time isn't entered
MsgBox "The time is now " & Time$
```

Phân tích

Sau khi định nghĩa hai biến Variant trong dòng 1 và 2 để giữ các giá trị ngày giờ được nhập vào bởi người dùng, chương trình sẽ hiển thị ngày trong dòng 3 và hỏi người dùng có muốn thay đổi ngày trong dòng 4 không. Dòng 5 sử dụng hàm IsDate(), bạn đã học trong bài trước, để kiểm tra người dùng đã nhập một giá trị ngày đúng. Nếu người dùng không nhập vào một ngày hợp lệ, chương trình sẽ nhảy qua dòng 6 và dòng 8 sẽ hiển thị xác lập ngày hiện hành.

Ghi chú

Visual Basic không chấp nhận bất kỳ giá trị ngày không hợp lệ nào. Nếu người dùng nhập vào 2–30–96, hàm IsDate() trong dòng 5 sẽ biết rằng tháng 02 không có 30 ngày và sẽ không xem đó là ngày hợp lệ.

Các dòng 9 đến 14 thi hành thủ tục tương tự với giờ.

XÁC ĐỊNH THỜI GIAN TRÔI QUA

Khái niệm

Hàm Timer() trong Visual Basic cho phép bạn tính toán thời gian trôi qua giữa hai thời điểm. Bằng cách sử dụng Timer(), bạn có thể nhận biết có bao nhiêu giây đã trôi qua từ lúc nửa đêm.

Hàm Timer() trả lại số giây đã trôi qua từ 12 giờ đêm. Timer() không yêu cầu đối số; do vậy, Visual Basic bỏ cặp ngoặc đơn của Timer() nếu bạn nhập chúng, nhưng tập sách này vẫn sử dụng cặp dấu () như một chuẩn, để nhắc bạn rằng Timer() là hàm chứ không phải lệnh.

Giả sử bây giờ là 11:40 p.m., lệnh sau sẽ lưu giá trị 85206.9 vào biến TMid:

```
TMid = Timer()
```

Timer() trả lại một giá trị chính xác đơn. Nếu bạn chia 85206.9 cho 60, bạn sẽ có số phút kể từ 12 giờ đêm, và nếu bạn chia cho 60 một lần nữa, bạn sẽ có số giờ (nhỏ hơn 24) kể từ 12 giờ đêm.

Lưu ý

Hàm Timer() hoàn toàn khác so với điều khiển bộ định giờ (Timer) trên hộp công cụ trong Chương 10.

Việc có thể xác định số giây tính từ 12 giờ đêm sẽ giúp một lập trình viên Visual Basic điều gì? Timer() là một biến cố thời gian hoàn hảo. Giả sử bạn hỏi người dùng một câu hỏi và muốn xác định người dùng mất bao nhiêu thời gian để trả lời. Đầu tiên, hãy lưu giá trị Timer() trước khi hỏi người dùng; sau đó, trừ giá trị đó với giá trị Timer có được sau khi người dùng trả lời. Sự khác biệt giữa hai giá trị Timer() là số giây người dùng cần để trả lời câu hỏi.

Tóm tắt

Ví dụ 14.3 chứa mã lệnh hỏi người dùng thời gian và tính số giây người dùng cần để trả lời câu hỏi đó.

Củng cố

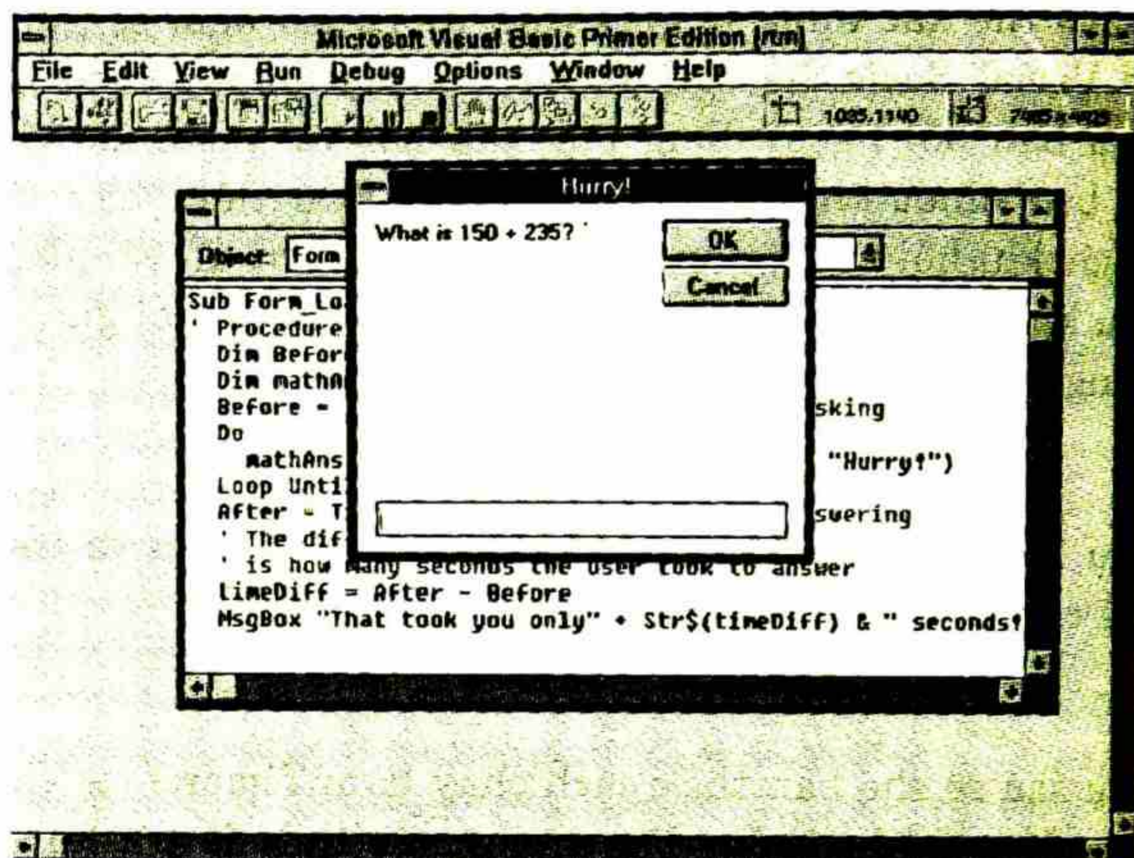
Ví dụ sau sẽ cho bạn thấy cách chạy hàm Timer() và Visual Basic sẽ kiểm tra đồng hồ hệ thống bên trong máy tính để trả lại số giây kể từ 12 giờ đêm.

Ví dụ 14.3. *Kiểm tra tốc độ xử lý của người dùng.*

```
Dim Before, After, timeDiff As Variant
Dim mathAns As String
Before = Timer ' Save the time before asking
Do
mathAns = InputBox("What is 150 + 235?", "Hurry!")
Loop Until Val(mathAns) = 285
After = Timer ' Save the time after answering
' The difference between the time values
' is how many seconds the user took to answer
timeDiff = After – Before
MsgBox "That took you only" + Str$(timeDiff) & " seconds!"
```

Kết quả xuất ra

Hình 14.1 hiển thị hộp nhận dữ liệu đợi câu trả lời của người dùng.



Hình 14.1. *Kiểm tra khả năng xử lý của người dùng.*

Phân tích

Dòng 3 lưu xác lập Timer() trước khi hộp nhận dữ liệu (Input Box) đợi câu trả lời của người dùng. Vòng lặp Do từ dòng 4 đến 6 kế đó duy trì việc đợi câu trả lời cho đến khi người dùng nhập vào giá trị thích hợp là 285. Ngay khi người dùng nhập vào câu trả lời đúng, dòng 7 lưu giá trị Timer() lúc đó. Sau đó, dòng 10 sẽ tính toán thời gian chênh lệch giữa 2 giá trị để xác định số giây mất cho câu trả lời.

GIÁ TRỊ NGÀY VÀ THỜI GIAN NỐI TIẾP

Khái niệm

Visual Basic hỗ trợ việc lưu các giá trị ngày và thời gian nối tiếp nhau. Giá trị chuỗi là một số được lưu trong kiểu VarType 7 (kiểu dữ liệu ngày, như bạn đã học trong bài trước). Số nối tiếp cho phép bạn chia ngày thành giá trị ngày, tháng, năm và chia thời gian thành giá trị giờ, phút và giây.

Khái niệm mới

Byte là một ký tự trong bộ nhớ.

Tất cả hàm ngày và thời gian đều làm việc như các giá trị nối tiếp (serial). Giá trị serial là nội dung mô tả bên trong một giá trị ngày giờ, được lưu theo kiểu VarType 7 hay kiểu dữ liệu Variant. Visual Basic thực sự lưu giá trị này dưới dạng giá trị chính xác kép để đảm bảo lưu đầy đủ ngày giờ. Visual Basic sẽ dùng 8 byte bộ nhớ cho việc lưu trữ các giá trị chính xác kép, đủ chỗ để giữ giá trị ngày giờ kết hợp.

Những hàm sau được giải thích trong phần này:

- DateSerial()
- DateValue()
- TimeSerial()
- TimeValue()
- Day()
- Month()
- Year()

Hết thảy hàm này đều đổi đối số của chúng thành giá trị ngày nối tiếp. Sau đó có thể sử dụng giá trị ngày nối tiếp này để quản lý và sửa đổi từng phần các giá trị ngày giờ xác định. Sau đây là dạng thức hàm DateSerial():

DateSerial(*Year, Month, Day*)

Trong đó, *Year* là số nguyên chỉ năm (hoặc từ 00 đến 99 cho 1900 tới 1999, hoặc năm có 4 ký số) hay biểu thức, *Month* là số nguyên chỉ tháng (1 tới 12) hay biểu thức, và *Day* là số nguyên chỉ ngày (1 tới 31) hay biểu thức. Nếu dùng biểu thức cho một đối số nguyên bất kỳ, bạn có thể xác định năm, tháng hay ngày bằng một giá trị.

Hai phép gọi hàm DateSerial() sau trả về cùng giá trị:

```
d = DateSerial(1990, 10, 6)
```

và

```
d = DateSerial(1980+10, 12-2, 1+5)
```

Hàm DateSerial() bảo đảm rằng các đối số ngày của bạn không vượt khỏi phạm vi. Ví dụ, 1992 là năm nhuận, nên tháng 02 năm 1992 có 29 ngày. Tuy nhiên, việc gọi hàm DateSerial() sau đưa ra một ngày không hợp lệ bởi vì tháng 02, thậm chí với cả năm nhuận, cũng không thể có 30 ngày:

```
d = DateSerial(1992, 2, 29+1)
```

Không có điều gì sai với lệnh gọi hàm này bởi vì DateSerial() sẽ tự điều chỉnh ngày để giữ ngày 1/3/1992, một ngày sau ngày cuối của tháng 2.

Hàm DateValue() tương tự hàm DateSerial() ngoại trừ việc hàm DateValue() chấp nhận đối số chuỗi, như dạng sau đây:

DateValue(stringDateExpression)

stringDateExpression phải là chuỗi mà Visual Basic nhìn nhận như một ngày (như lệnh Date$ đã mô tả trước đây trong chương và các ngày hợp lệ được người dùng nhập vào). Nếu bạn yêu cầu người dùng nhập vào từng phần giá trị của một ngày tại một thời điểm (yêu cầu về năm, sau đó tháng, cuối cùng là ngày), bạn có thể dùng hàm DateSerial() để chuyển các giá trị này thành một giá trị ngày nối tiếp lưu trong bộ nhớ. Nếu bạn yêu cầu người dùng nhập vào một ngày đầy đủ (gán vào biến chuỗi) như là **July 16, 1996**, hàm DateSerial() đổi chuỗi này thành dạng giá trị ngày serial.

Các hàm TimeSerial() và TimeValue() làm việc giống người bà con của chúng là DateSerial() và DateValue(). Nếu bạn có ba giá trị riêng biệt về thời gian trong ngày, hàm TimeSerial() sẽ đổi các giá trị này thành dạng thời gian lưu trong máy (Variant hay VarType 7). Sau đây là dạng thức hàm TimeSerial():

TimeSerial(*Hour, Minute, Second*)

Hàm TimeSerial() chấp nhận biểu thức trong các đối số và điều chỉnh biểu thức này khi cần thiết, ngay khi hàm DateSerial() thực hiện.

Nếu bạn có chuỗi tương ứng một giá trị thời gian (có thể người dùng đã nhập thời gian), hàm TimeValue() đổi chuỗi đó thành giá trị thời gian với dạng sau :

TimeValue(stringTimeExpression)

Các hàm Day(), Month(), và Year() đổi đối số ngày (theo kiểu dữ liệu Variant hay VarType 7) thành giá trị ngày, giá trị tháng, hay giá trị năm. Ba hàm này đơn giản thôi. Sau đây là các dạng thức chúng:

Day(DateArgument)

Month(DateArgument)

Year(DateArgument)

Bạn thường chuyển ngày hiện tại (được tìm thấy với hàm Now()) thành ngày, tháng, năm bằng hàm Day(), Month(), và Year() như dưới đây:

```
d = Day(Now())
m = Month(Now())
y = Year(Now())
```

Giá trị ngày trong tuần, tháng, năm của ngày hiện hành được lưu riêng trong ba biến d, m, và y.

Tóm tắt

Ví dụ 14.4 mô tả đoạn mã lệnh ngắn yêu cầu người dùng nhập một ngày, sử dụng hàm IsDate() để đảm bảo rằng ngày nhập là đúng, và sau đó sử dụng các hàm Month(), Day(), và Year() để chia ngày thành các phần riêng biệt.

Củng cố

Giá trị ngày nối tiếp lưu trong bộ nhớ mà Visual Basic sử dụng cho tất cả giá trị ngày giờ cho phép bạn sử dụng các hàm khác nhau để chia thành từng phần hoặc nối các tổ hợp ngày giờ khác nhau lại.

Ví dụ 14.4. *Làm việc với các giá trị ngày nối tiếp.*

```
Dim UserDate As Variant
Dim DStr As String

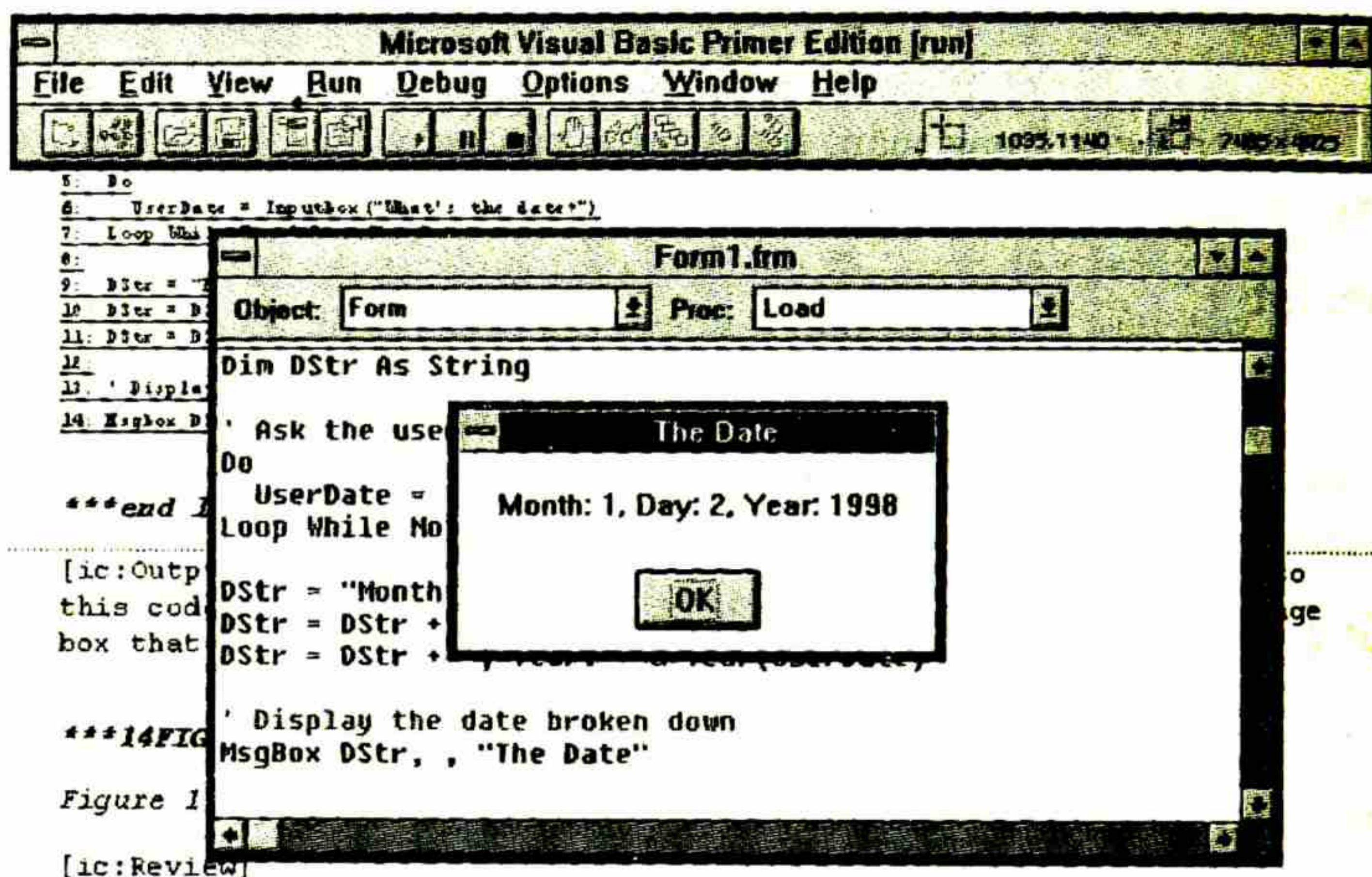
' Ask the user for a date
Do
UserDate = InputBox("What's the date?")
Loop While Not IsDate(UserDate)

DStr = "Month: " & Month(UserDate)
DStr = DStr + ", Day: " & Day(UserDate)
DStr = DStr + ", Year: " & Year(UserDate)

' Display the date broken down
MsgBox DStr
```

Kết quả xuất ra

Giả sử người dùng nhập giá trị **1/2/98** cho ngày để trả lời lệnh gọi hàm InputBox(), Hình 14.2 mô tả hộp thông báo trình bày các thành phần khác nhau của ngày đã nhập.



Hình 14.2. *Thông qua các hàm, bạn có thể chia nhỏ 1 ngày.*

Củng cố

Dòng 1 và 2 định nghĩa 2 biến chương trình. Biến Variant UserDate sẽ giữ giá trị ngày được nhập vào bởi người dùng, và biến DStr sẽ giữ chuỗi trong hộp thông báo.

Các dòng 5 đến 7 sẽ lặp cho đến khi người dùng nhập vào giá trị ngày thích hợp. Vòng lặp sẽ tiếp tục khi hàm IsDate() cho biết giá trị ngày của người dùng là không hợp lệ. Vòng lặp sẽ thoát bằng việc hỏi người dùng về ngày khi dòng 7 xác định rằng ngày hợp lệ.

Các dòng 9 đến 11 sẽ xây dựng một chuỗi khá phức tạp được hiển thị bởi hàm message box ở dòng 14. Các dòng nối nhiều chuỗi với nhau, xây dựng trên biến Dstr. Các hàm Month(), Day(), và Year() đều trích giá trị tháng, ngày, năm từ ngày người dùng nhập vào để hiển thị các giá trị khác nhau trong hộp thông báo, như được mô tả trong Hình 14.2.

ĐỊNH DẠNG VỚI HÀM Format()

Khái niệm mới

Giá trị logic là kết quả một quan hệ có giá trị điều khiển hoặc True hoặc False.

Khái niệm

Hàm Format() định dạng các giá trị số và logic để hiển thị dữ liệu ở dạng chính xác phù hợp yêu cầu của bạn. Cho đến lúc này, bạn không thể bảo đảm rằng các giá trị currency hiển thị với hai chữ số thập phân, nhưng hàm Format() cho phép bạn định dạng currency và tất cả các dạng số khác theo cách bạn cần.

Visual Basic không thể hiểu ý bạn, cho nên nó không biết cách bạn muốn trình bày các số trong trình ứng dụng. Mặc dù đôi khi Visual Basic không hiển thị hoặc hiển thị một hay nhiều chữ số thập phân cho các giá trị currency, bạn luôn muốn các giá trị currency này được hiển thị với hai chữ số thập phân.

Hai hàm định dạng là Format$() và Format() chỉ khác biệt về kiểu dữ liệu trả lại. Format$() trả về một chuỗi và Format() trả về kiểu dữ liệu Variant. Sau đây là dạng thức hàm Format():

```
Format$(Expression, FormatStr)
```

Thông thường, bạn sẽ gán kết quả hàm Format$() cho các biến và những điều khiển khác. Nhìn chung, bạn thực hiện tất cả tính toán cần thiết trên các giá trị số trước khi định dạng những giá trị này. Sau khi thực hiện tính toán cuối cùng, lúc đó bạn sẽ định dạng các giá trị thành kiểu dữ liệu chuỗi (hay Variant) và hiển thị kết quả trả lời khi cần.

Expression có thể là biến, biểu thức, hay hằng số. Hàm FormatStr phải là một giá trị trong Bảng 14.1.

Ghi chú

Visual Basic có nhiều dạng chuỗi thêm vào các dạng trong Bảng 14.1. Thậm chí bạn có thể định nghĩa các hàm định dạng chuỗi của riêng bạn, tuy nhiên tập sách này không đi sâu vào điều đó.

Khái niệm mới

Dấu phân cách hàng ngàn là một dấu chấm thập phân hay dấu phẩy bên trong các số trên 999.

Bảng 14.1. *Các giá trị FormatStr cố định*

FormatStr	Diễn giải
"Currency"	Bảo đảm rằng dấu $, sẽ xuất hiện trước giá trị được định dạng theo sau bởi các dấu phân cách hàng ngàn (xác lập country sẽ xác định dấu phân cách hàng ngàn là dấu phẩy hay dấu chấm), và hai vị trí thập phân. Visual Basic sẽ hiển thị các giá trị âm trong dấu ngoặc đơn.
"Fixed"	Hiển thị ít nhất một số trước và hai số sau dấu chấm thập phân không có dấu phân cách hàng ngàn.
"General Number"	Hiển thị số không có dấu phân cách hàng ngàn.
"Medium Time"	Hiển thị thời gian ở dạng 12 giờ và chỉ báo AM hay PM.
"On/Off"	Hiển thị On nếu giá trị chứa một số khác 0 hay có giá trị True và hiển thị Off nếu giá trị chứa số 0 hay giá trị False. Những giá trị này thích hợp với một trình bày bên trong xác định của các giá trị máy tính gọi là số nhị phân.
"Percent"	Hiển thị số được nhân với 100, và thêm dấu % ở bên phải số đó.
"Scientific"	Hiển thị các số ở dạng khoa học (notation).
"Short Time"	Hiển thị thời gian ở dạng 24 giờ.
"True/False"	Hiển thị True nếu giá trị có một số khác 0 hay giá trị True, và hiển thị False nếu giá trị bằng 0 hay một giá trị False.
"Yes/No"	Hiển thị Yes nếu giá trị có một số khác 0 hay giá trị True, và hiển thị No nếu giá trị bằng 0 hay một giá trị False.

Ít khi bạn cần các mã lệnh định dạng. Nếu các dạng thức được định nghĩa trước từ Bảng 14.1 không phù hợp dạng thức bạn cần, bạn có thể định nghĩa dạng thức riêng cho bạn bằng việc sử dụng các mã lệnh định dạng. Trong bài này ít nhất có hai lần sử dụng nó. May cho bạn là khi định nghĩa các dạng thức riêng, hầu như bạn sẽ sử dụng tổ hợp dấu # và 0 thích hợp để định dạng các giá trị.

Mỗi dấu # trong phần định dạng cho biết vị trí một số sẽ hiển thị, và số 0 cho biết bạn muốn hoặc số 0 ở đầu hoặc số 0 ở cuối. Phép gán sau sẽ hiển thị giá trị của Weight với ba vị trí thập phân:

```
lblMeas.Caption = Format$(Weight, "######.000")
```

Bạn có thể yêu cầu không hiển thị chữ số thập phân nào bằng cách định dạng giá trị fractional như là Weight, và Visual Basic sẽ làm tròn số khi cần để phù hợp với dạng đích của nó. Phép gán sau sẽ hiển thị Weight không có số thập phân nào:

```
lblMeas.Caption = Format$(Weight, "######")
```

Tóm tắt

Ví dụ 14.5 trình bày một số lệnh gọi hàm định dạng đổi các giá trị số và logic thành những kiểu dữ liệu Variant được định dạng để bạn có thể hiển thị.

Củng cố

Có hai hàm định dạng: Format$() trả về giá trị chuỗi và Format() trả về giá trị kiểu Variant. Hàm Format$() định dạng các giá trị theo cách bạn muốn. Các chuỗi định dạng được định nghĩa trước để giúp hiển thị dễ dàng những giá trị số và logic theo dạng mong muốn.

Ví dụ 14.5. *Định dạng các giá trị số và logic.*

```
Dim FormValue As String

' Change 12345.678 to $12,345.68
FormValue1 = Format$(12345.678, "Currency")

' Change 12345678 to 12345.68
FormValue2 = Format$(12345.678, "Fixes")

' Change .52 to 52.00%
FormValue3 = Format$(.52, "Percent")

' Change 1 to Yes
FormValue4 = Format$(1, "Yes/No")
```

```
14:
15: ' Change 0 to No
16: FormValue5 = Format$(0, "Yes/No")
17:
18: ' Change 1 to True
19: FormValue6 = Format$(1, "True/False")
20:
21: ' Change 0 to False
22: FormValue7 = Format$(0, "True/False")
```

Phân tích

Dòng 4 đổi giá trị số chính xác đơn thập phân thành chuỗi currency được định dạng. Nếu bạn đã hiển thị FormValue1 trong đề mục của nhãn (label), đề mục sẽ chỉ giá trị $12,345.68. Dòng 7 cũng định dạng số đó với một giá trị không currency với hai chữ số thập phân. Dòng 10 sẽ thay đổi một số fractional thành giá trị phần trăm thập phân tương ứng của nó và thêm dấu % sau phần trăm được định dạng để chỉ dạng thành phần phần trăm. Dòng 13 sẽ thay đổi giá trị khác 0 là 1 thành Yes. Bạn có thể định dạng một kết quả khác 0 hay quan hệ bất kỳ thành giá trị Yes, No, True, hay False, như trong các dòng còn lại của chương trình. Vì thế các giá trị sẽ xuất hiện chính xác theo cách bạn muốn.

Bài tập

Kiến thức tổng quát

1. Đúng hay Sai: Hàm Now() trả lại thông tin về ngày giờ hiện hành.
2. Hàm ngày giờ lấy giá trị của chúng ở đâu?
3. Nét khác biệt giữa giờ tính theo 12 giờ và 24 giờ?
4. Đúng hay Sai: 9:23 có thể là thời gian tính theo 12 giờ hoặc theo 24 giờ.
5. Đúng hay Sai: 9:23 a.m có thể là thời gian tính theo 12 giờ hoặc theo 24 giờ.

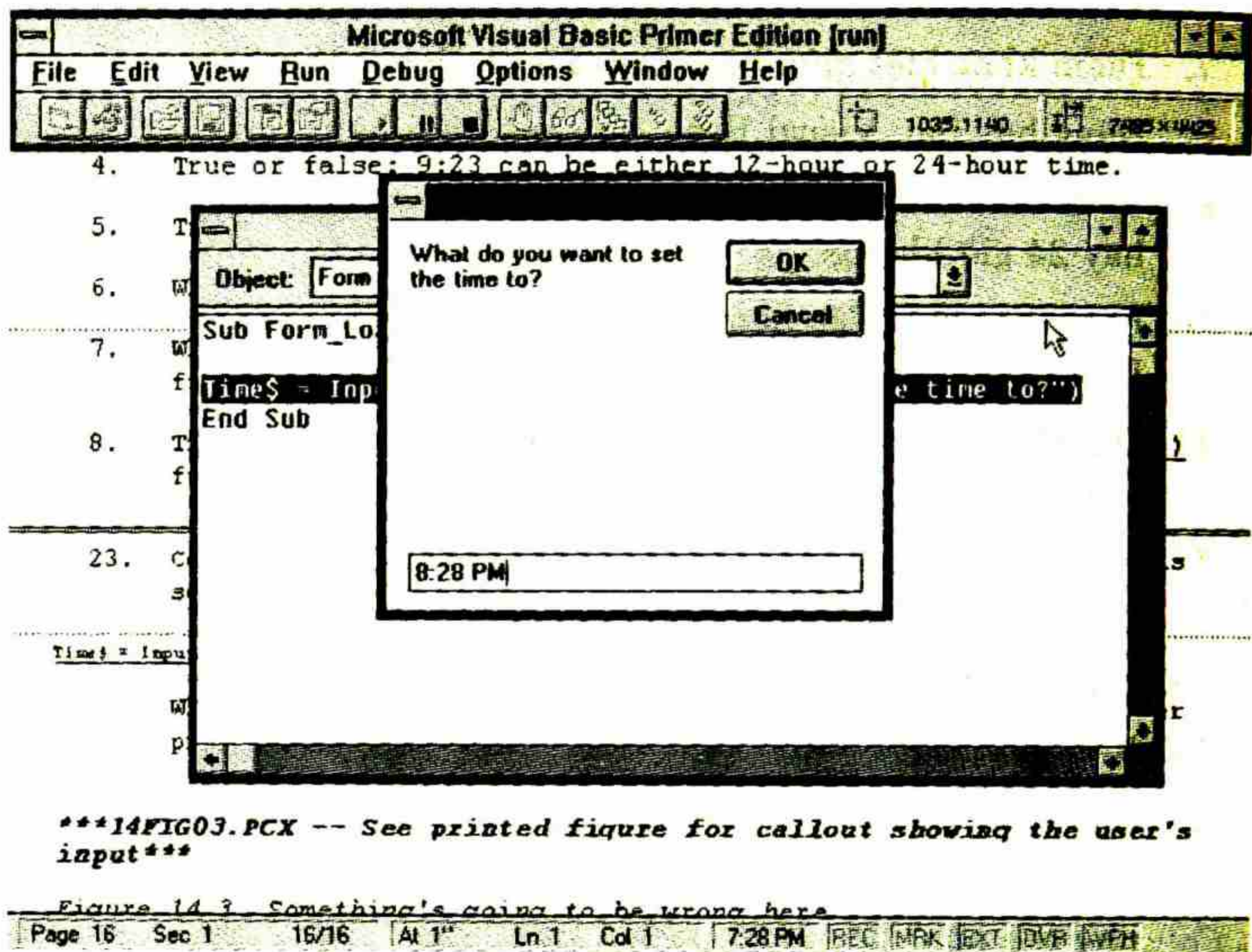
6. Giá trị thời gian 24 giờ của 7:54 p.m là gì?
7. Điểm khác biệt giữa hàm Date$() và Date() là gì?
8. Đúng hay Sai: Visual Basic hỗ trợ cả hai hàm Now() và Now$().
9. Hàm Now() có trả về giá trị thời gian sử dụng đồng hồ 12 giờ hay 24 giờ không?
10. Đúng hay Sai: Các hàm ngày giờ bỏ qua những xác lập ngày giờ International.
11. Đúng hay Sai: Hàm Date() và Time() ấn định ngày giờ máy tính của bạn.
12. Nên dùng hàm nào, Date() hay Date$(), nếu bạn nghĩ rằng chương trình máy tính của bạn được sử dụng trong năm 2000?
13. Hãy mô tả mục đích của hàm Timer().
14. *Byte* là gì?
15. Kiểu dữ liệu nào được đòi hỏi cho một giá trị ngày nối tiếp?
16. Hàm TimeValue() quan hệ với hàm Hour(), Minute(), và Second() như thế nào?
17. Giá trị logic là gì?
18. Những hàm nào định dạng số xuất ra cho bạn?
19. Điểm khác biệt giữa Format() và Format$() là gì?
20. Dấu phân cách hàng ngàn là gì?
21. Điểm khác biệt giữa các chuỗi định dạng cố định "Fixed" và "Currency" là gì?

Tìm lỗi kỹ thuật

22. Hãy xem xét dữ liệu nhập vào của người dùng trong Hình 14.3. Chương trình đang ấn định thời gian vào hộp nhận dữ liệu này.

```
Time$ = InputBox("What do you want to set the time to?")
```

Tại sao bạn nghĩ rằng Visual Basic sẽ đưa ra một thông báo lỗi (ngay khi người dùng nhấn OK trong hộp nhận dữ liệu)?



Hình 14.3. *Có điều gì đó không ổn ở đây.*

Phần nâng cao

Hãy viết mã lệnh hỏi người dùng thời gian vào làm việc và sau đó hỏi thời gian người dùng sẽ thoát ra. Hiển thị ba nhãn cho biết tổng số giây, phút và giờ đã làm việc.

Bài thực hành 7

Hàm và ngày tháng

Tóm tắt

Chương này trình bày hết thảy hàm số và chuỗi có sẵn trong Visual Basic. Những hàm này giúp bạn tiết kiệm thời gian lập trình bởi vì không phải mất thời gian để viết mã lệnh thi hành các nhiệm vụ chung đó. Một vài hàm số giúp đổi số, cũng như hàm khoa học và toán học. Hàm chuỗi phân tích và thay đổi chuỗi cho phù hợp dạng bạn cần.

Trong chương này, bạn đã nắm bắt:

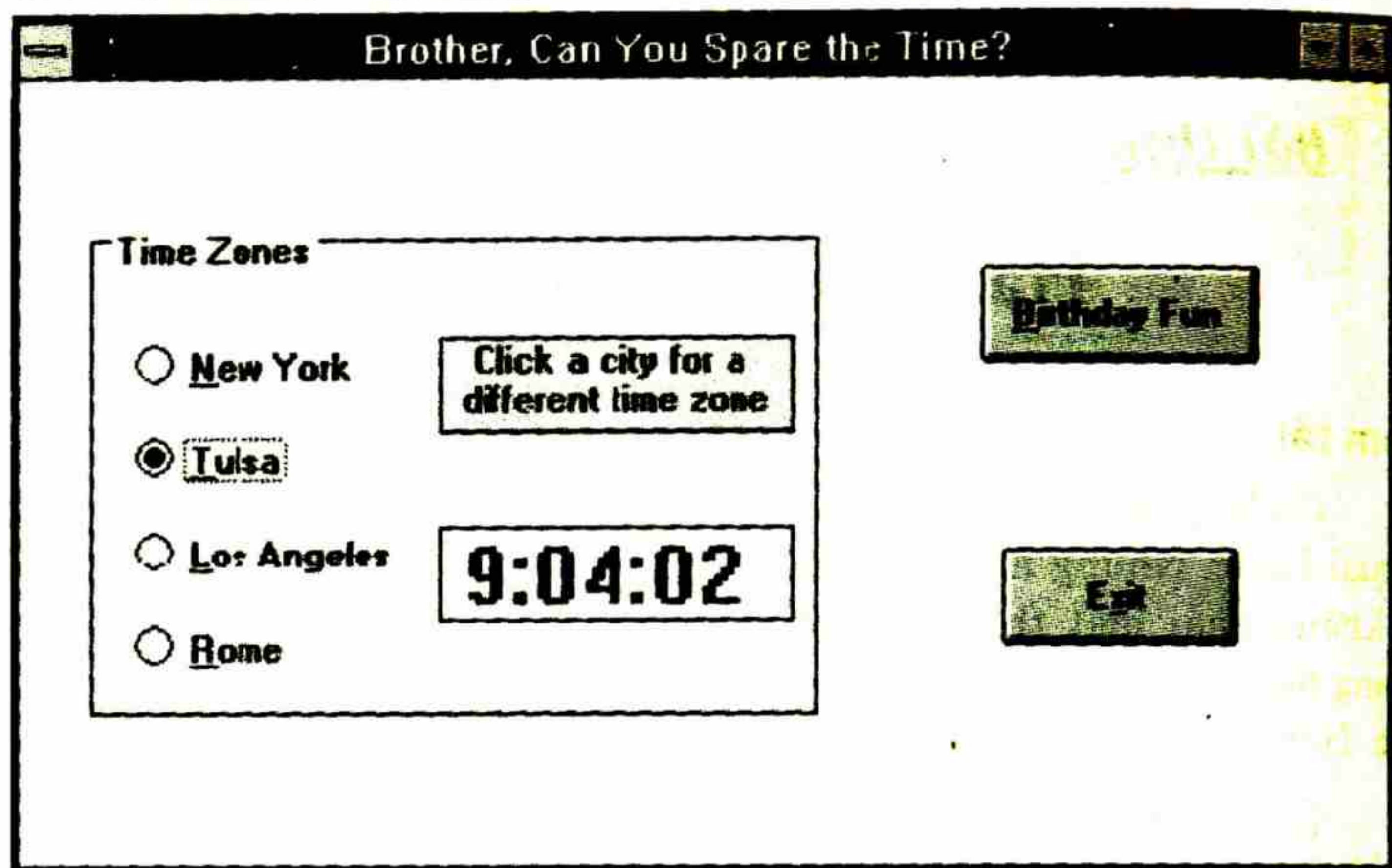
- Lý do tại sao hàm đẩy nhanh quá trình thiết kế chương trình
- Khi nào cần sử dụng một trong ba hàm chuyển đổi số nguyên
- Visual Basic cung cấp các hàm toán học và khoa học nào
- Hàm Date() sẽ sử dụng khi bạn cần lấy hay tính giá trị ngày giờ
- Hàm Format() cho phép bạn xác định dạng dữ liệu xuất ra như thế nào

Mô tả chương trình

Hình P7.1 minh họa ứng dụng PROJECT7.VBP ngay khi bạn nạp và chạy chương trình. Mẫu biểu của project có một khung với bốn nút tùy chọn và hai nút lệnh.

Ghi chú

Quá trình tính toán múi giờ trong bài thực hành này giả sử rằng bạn cư ngụ trong múi giờ chuẩn trung tâm. Dĩ nhiên, có thể bạn không sống trong vùng thời gian CST, nhưng chương trình phải giả sử một số loại vùng trên cơ sở đó sẽ thay đổi quá trình tính toán múi giờ.



Hình P7.1. *Màn hình đang mở của project.*

Nút tùy chọn chương trình chọn một thành phố khác thời gian địa phương sẽ hiển thị trong nhãn thời gian lớn trong khung. Thời gian chỉ cập nhật khi lần đầu tiên bạn chạy chương trình và khi bạn nhấp nút tùy chọn.

Nút lệnh Birthday Fun sẽ tính năm bạn về hưu để thực hiện một chuyến du lịch hai năm bạn từng mơ tới.

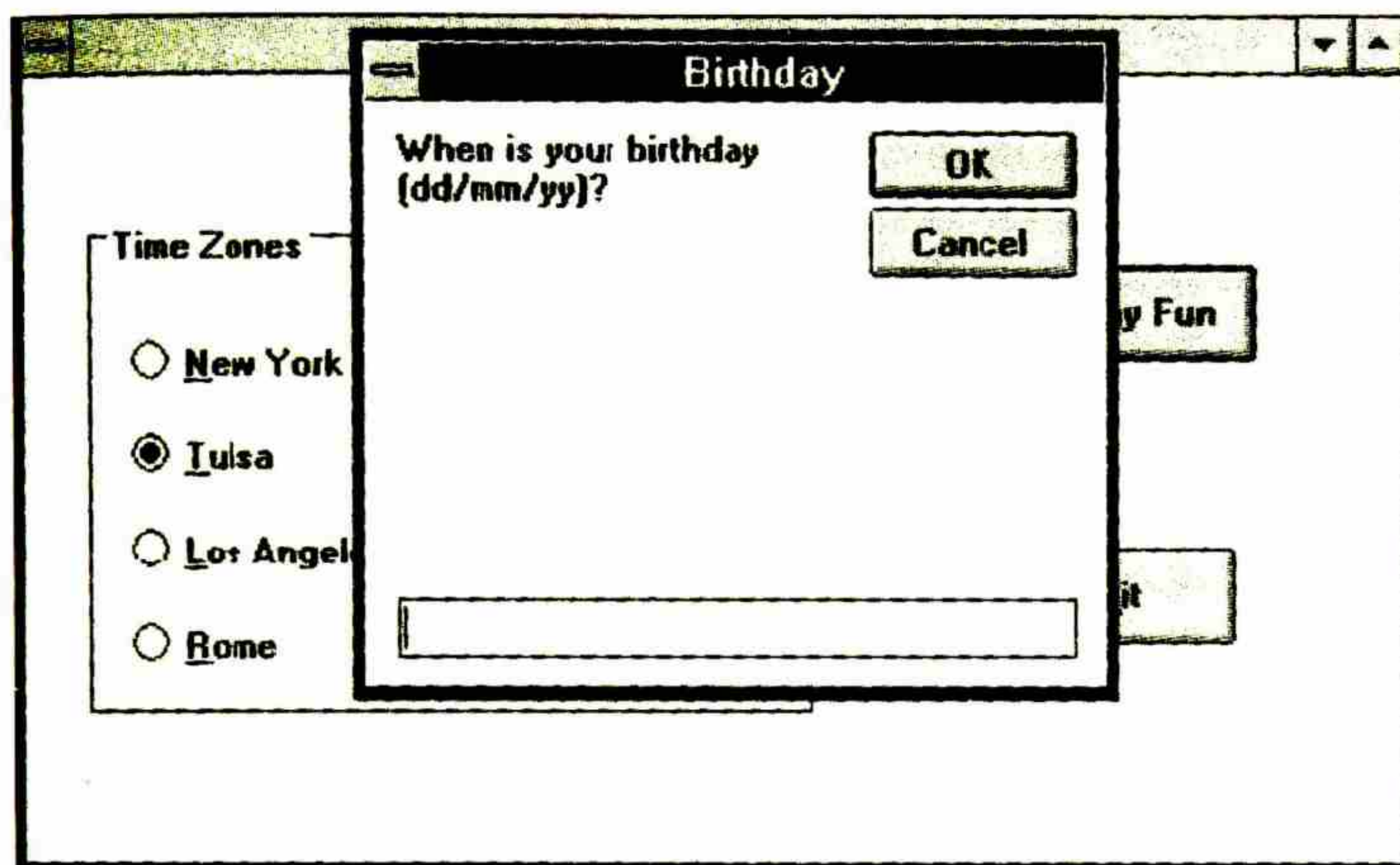
Hoạt động chương trình

Hãy thử điều này: Nhấp các nút tùy chọn để lựa chọn những thành phố khác nhau và xem múi giờ thay đổi. Những nút tùy chọn tạo thành một dãy thành phần trong mảng điều khiển nút tùy chọn, và các giá trị chỉ mục cho nút tùy chọn sẽ giúp mã lệnh ở dưới các nút (được mô tả trong phần kế tiếp) xác định múi giờ nào để tính.

Mách nước

Trước khi xem mã lệnh của bài thực hành này, hãy thử xác định cách bạn cộng hay trừ giá trị múi giờ hiện hành, bằng cách dùng mã lệnh Visual Basic, để tính một múi giờ khác. Bài thực hành này sẽ giúp bạn nhiều điều và cô đọng phần thảo luận hàm thời gian của bài trước.

Nhấp nút lệnh Birthday Fun và bạn sẽ thấy hộp nhận dữ liệu được mô tả trong Hình P7.2 hỏi ngày sinh của người dùng. Một hộp thông báo kế tiếp sẽ báo cho người dùng năm có thể nghỉ hưu, dựa trên tuổi hưu là 65.



Hình P7.2. *Một hộp nhận dữ liệu nhận ngày sinh người dùng.*

Thủ tục thay đổi múi giờ

Ví dụ P7.1 trình bày thủ tục biến cố cho những nút tùy chọn múi giờ lưu trong mảng điều khiển optCity.

Ví dụ P7.1 *Xác định thời gian trong các múi giờ khác nhau.*

```
Sub optCity_Click (Index As Integer)
' Adds or subtracts a value from the time
' that matches the time zone selected
Dim DiffVal As Integer ' Hours to add or subtract

' Test the option button's index
Select Case Index
Case 0:
' NYC
DiffVal = 1
Case 1:
' Tulsa, the Central Standard Time default
DiffVal = 0
Case 2:
```

```
15: ' Los Angeles
16: DiffVal = -2
17: Case 3:
18: ' Rome
19: DiffVal = -7
20: End Select
21: lblTime.Caption = Format$(TimeSerial(Hour(Now) + DiffVal, Minute(Now),
Second(Now)), "Medium Time")
22: End Sub
```

Mô tả

1: Mảng nút tùy chọn mệnh danh optCity, cho nên tên thủ tục biến cố Click là optCity_Click(). Giá trị chỉ mục cũng được chuyển tới thủ tục biến cố nhờ đó thủ tục sẽ biết nút tùy chọn nào sẽ kích hoạt biến cố.

2: Chú thích giúp giải thích mục đích thủ tục biến cố.

3: Chú thích tiếp tục giúp giải thích mục đích thủ tục biến cố.

4: Định nghĩa một mảng lưu số giờ để cộng hay trừ giờ hiện hành thu được từ giá trị múi giờ mới.

5: Dòng trống phân chia định nghĩa biến với phần còn lại của mã lệnh.

6: Chú thích giúp giải thích mục đích lệnh Select Case kế tiếp.

7: Bắt đầu quá trình chọn mã lệnh chính để thi hành dựa trên chỉ số nút tùy chọn được chọn.

8: Trường hợp chọn nút tùy chọn đầu tiên (có tên nhãn là New York).

9: Chú thích mô tả việc chọn trường hợp này.

10: Cộng 1 giờ vào giá trị giờ hiện hành (giả sử là giờ chuẩn trung tâm).

11: Trường hợp chọn nút tùy chọn thứ hai (có tên nhãn là Tulsa).

12: Chú thích mô tả việc chọn trường hợp này.

13: Không thay đổi giá trị giờ chuẩn trung tâm hiện hành.

14: Trường hợp chọn nút tùy chọn thứ ba (có tên nhãn là Los Angeles).

15: Chú thích mô tả việc chọn trường hợp này.

16: Trừ giá trị giờ chuẩn trung tâm đi 2 giờ.

17: Trường hợp chọn nút tùy chọn thứ tư (có tên nhãn là Rome).

18: Chú thích mô tả việc chọn trường hợp này.

19: Trừ giá trị giờ chuẩn trung tâm hiện hành đi 7 giờ.

20: Kết thúc lệnh Select Case.

21: Cập nhật đề mục với giá trị thời gian được định dạng mới. Hàm TimeSerial() cho phép bạn xác định chỉ thay đổi giờ.

21: Với hàm TimeSerial(), bạn có thể truy xuất tới từng phần thời gian riêng biệt.

22: Kết thúc thủ tục biến cố.

Tính thời gian nghỉ hưu

Khi người dùng nhấp nút lệnh Birthday Fun, chương trình sẽ hỏi người dùng ngày sinh và tính năm dựa trên ngày sinh của người dùng. Ví dụ P7.2 trình bày thủ tục biến cố tính tuổi nghỉ hưu.

Ví dụ P7.2. *Tính năm nghỉ hưu.*

```
1: Sub cmdBirth_Click ()
2: ' Asks the user for a birthdate and determine retirement
3: Dim BDate As Variant
4: Dim BYear, RetYear As Integer
5: ' Get the birthday
6: Do
7: BDate = InputBox("When is your birthday (dd/mm/yy)?", "Birthday")
8: Loop Until IsDate(BDate) Or BDate = ""
9: If BDate <> "" Then ' If not Cancel
10: ' Filter out the birth year
11: BYear = Year(BDate)
12: ' Add the number of years to retirement
13: RetYear = BYear + 65
14: MsgBox "You can retire in" & Str$(RetYear)
15: End If
16: End Sub
```

Mô tả

1: Thuộc tính name của nút lệnh là cmdBirth, vì thế tên của thủ tục biến cố Click là cmdBirth_Click().

2: Chú thích giải thích mục đích thủ tục.

3: Định nghĩa một biến Variant sẽ lưu ngày sinh của người dùng.

4: Định nghĩa hai biến số nguyên. Một biến sẽ lưu năm sinh của người dùng, biến còn lại sẽ lưu năm nghỉ hưu.

5: Chú thích giải thích mục đích mã lệnh kế tiếp.

6: Bắt đầu vòng lặp nhận ngày hợp lệ.

7: Hỏi ngày sinh của người dùng.

8: Duy trì vòng lặp cho đến khi người dùng nhập vào một ngày hợp lệ hay khi nhấn Cancel.

9: Thực hiện thủ tục tính ngày nghỉ hưu nếu người dùng không nhấn Cancel.

10: Chú thích giải thích mục đích mã lệnh kế tiếp.

11: Tách năm trong ngày sinh và lưu năm đó.

11: Hàm Year() cho phép lấy năm từ một giá trị ngày.

12: Chú thích giải thích mục đích mã lệnh kế tiếp.

13: Cộng 65 với năm sinh để tìm năm nghỉ hưu.

14: Hiển thị thông báo năm nghỉ hưu

15: Kết thúc lệnh If.

16: Kết thúc thủ tục biến cố.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Chương kế tiếp sẽ giải thích cách thiết kế các chương trình phức tạp hơn bằng cách tách mã lệnh chung bạn cần viết trong nhiều thủ tục không biến cố riêng.

Chương VIII

Bài 15

Chương trình con

- **Giới thiệu về chương trình con**
- **Subroutine: thủ tục mã lệnh**
- **Module có phần mở rộng .BAS**
- **Thủ tục Function**

Bài này thiên về lý thuyết nhiều hơn những bài khác. Trước khi đi bước nữa trong lập trình Visual Basic, bạn phải làm chủ các kỹ thuật lập trình cần thiết khi viết các ứng dụng quy mô lớn.

GIỚI THIỆU VỀ CHƯƠNG TRÌNH CON

Khái niệm

Chia chương trình của bạn thành nhiều chương trình nhỏ nhưng hợp logic là điều có thể được. Các thủ tục con nhỏ hơn giúp cho việc lập trình và bảo trì sau này của bạn dễ dàng hơn.

Trong nhiều ngôn ngữ lập trình truyền thống như COBOL và FORTRAN, chương trình giống như một cuốn sách dài không có chương nào: Mã lệnh nối tiếp liên tục và chiều dài của chương trình được phóng đại bằng vẻ mặt khó chịu của các lập trình viên đang cố tìm ra lỗi mã lệnh. Bạn đã biết rằng chương trình Visual Basic gồm có nhiều danh sách chương trình dài.

- Một mẫu biểu với nhiều điều khiển hoạt động như giao diện người dùng và nền của chương trình.
- Một thủ tục (General) được tìm thấy trong danh sách cuộn xuống Object của cửa sổ Code.

- Các thủ tục biến cố buộc những điều khiển với nhau và bổ sung cơ chế định hướng và tính toán rõ ràng vào ứng dụng.
- Một tập tin CONSTANT.BAS cung cấp các tên hằng được mã lệnh của bạn sử dụng.

Ghi chú

Thủ tục (General) trong cửa sổ Code của chương trình bất kỳ thường gọi là vùng khai báo.

Bài này sẽ mở rộng mục đích của thủ tục (General) đồng thời giải thích cách thêm các loại thủ tục khác nhau vào chương trình. Bây giờ bạn đã biết hai loại thủ tục: thủ tục biến cố sẽ thi hành khi các biến cố xảy ra, và thủ tục (General). Thủ tục (General) thực sự không phải là thủ tục có thể thi hành giống như thủ tục biến cố. Câu lệnh duy nhất có thể đặt trong thủ tục (General) là lệnh định nghĩa dữ liệu như lệnh Dim và lệnh tùy chọn Option như sau:

```
Option Base 1
```

Và

```
Option Explicit
```

Nhiều lập trình viên Visual Basic không lo lắng về việc sử dụng lệnh Option Base 1, vốn xác định tất cả chỉ số mảng sẽ bắt đầu bằng 1 thay vì bằng 0 (mặc định). Thay vì sử dụng Option Base 1, đa số lập trình viên bỏ qua chỉ số 0 đầu tiên và hoạt động như các chỉ số trong mảng bắt đầu là 1.

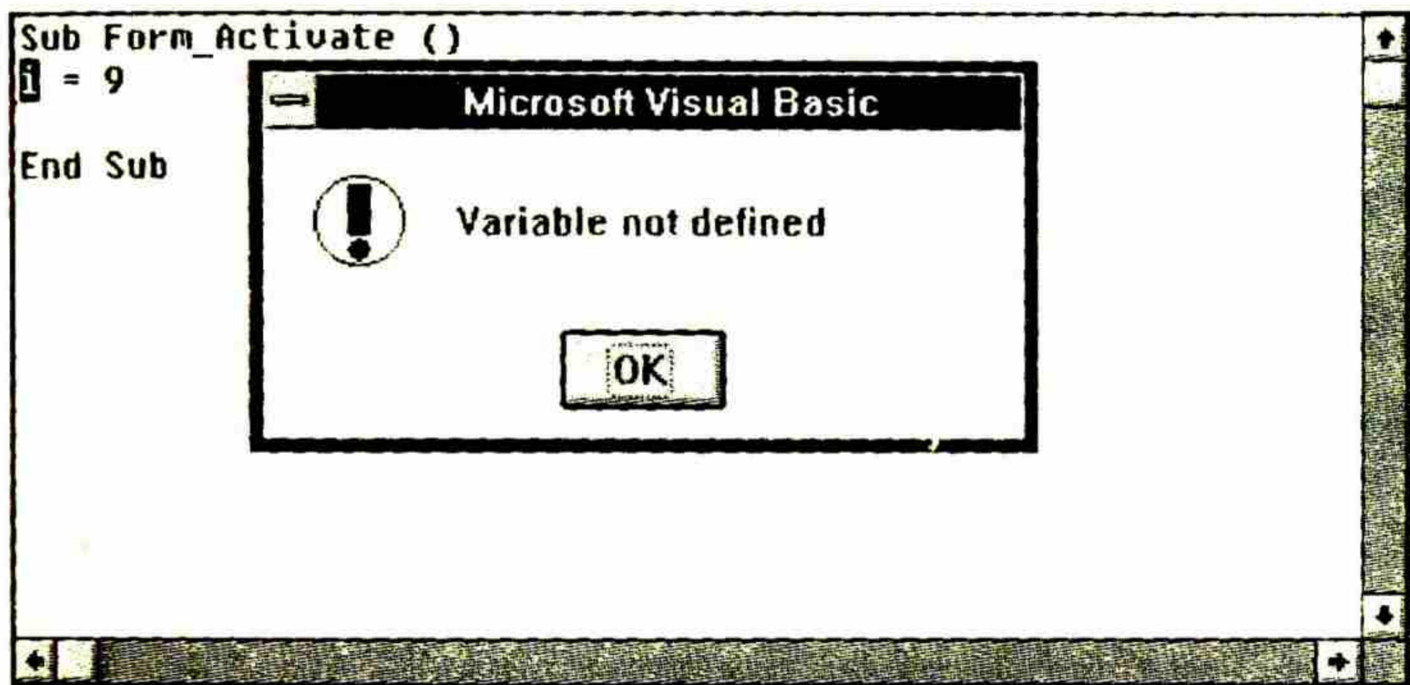
Lệnh Option Explicit là một lệnh hữu dụng báo cho Visual Basic tìm bất kỳ biến không được định nghĩa và đưa ra lỗi nếu Visual Basic tìm thấy. Bằng cách yêu cầu bạn định nghĩa rõ ràng tất cả các biến trước khi sử dụng chúng, lỗi tên biến không bao giờ xảy ra làm đảo ngược kết quả.

Giả sử chương trình của bạn không có lệnh Option Explicit và bạn nhập các lệnh sau vào trong một thủ tục biến cố:

```
Dim Sales As Currency
Sale = 294.43
lblOut.Caption = Sales ' Outputs zero
```

Visual Basic xuất 0 ra nhãn bởi vì Sales bằng 0, và biến mà bạn nghĩ tên là Sales, bạn lại tình cờ đặt tên là Sale, giữ giá trị 294.43, là biến mà người dùng sẽ không nhìn thấy.

Nếu bạn đã thêm lệnh Option Explicit trong thủ tục (General) (nơi duy nhất bạn có thể đặt các câu lệnh Option), Visual Basic sẽ hiển thị hộp thông báo lỗi như trong Hình 15.1, bởi vì Visual Basic suy luận đúng rằng bạn chưa định nghĩa biến Sale nhưng lại cố lưu một giá trị vào Sale.



Hình 15.1. *Định nghĩa tất cả biến trước khi sử dụng chúng, bạn sẽ nhận lỗi này.*

Khái niệm mới

Việc khai báo biến, trong Visual Basic, giống như định nghĩa biến.

Ghi chú

Menu Options Environment sẽ hiển thị một danh sách tùy chọn chứa tùy chọn Require Variable Declaration có thể định là True. Nếu một thủ tục (General) của chương trình bao gồm lệnh Option Explicit, chương trình đòi hỏi tất cả biến được định nghĩa trước khi sử dụng chúng, không phụ thuộc vào giá trị định trong menu Options Environment.

Bạn có thể ấn định một số thủ tục bổ sung trong chương trình của bạn. Tên chung cho những thủ tục này là *chương trình con* (subprogram). Các thủ tục này, chẳng hạn như thủ tục biến cố, giống những phiên bản nhỏ của một chương trình Visual Basic. Tất cả thủ tục xây dựng các khối mã lệnh cho cả ứng dụng bởi vì một ứng dụng đầy đủ bao gồm tất cả thủ tục làm việc với nhau.

Có hai loại chương trình con trong Visual Basic: thủ tục Subroutine và thủ tục Function. Thông thường, chúng ta viết tắt thủ tục Subroutine là *subroutine* và thủ tục Function là *function*. Điều này có thể gây nhầm lẫn do có hàm xây dựng sẵn như Int() mà bạn đã làm quen ở bài trước.

Ôn lại

Phần này liên quan đến các thuật ngữ bạn sẽ đọc và sử dụng trong phần còn lại của cuốn sách. Các chương trình Visual Basic thực ra chỉ là những đoạn mã lệnh nhỏ gọi là *subprogram*. Phần còn lại của bài này sẽ giải thích nhiều hơn về việc chia chương trình Visual Basic cũng như giải thích cách lưu các chương trình con trong tập tin ngoài để cho nhiều ứng dụng Visual Basic có thể chia sẻ.

Subroutine: THỦ TỤC MÃ LỆNH

Khái niệm

Một thủ tục con luôn luôn bắt đầu bằng lệnh Sub và kết thúc bằng lệnh End Sub. Thủ tục Subroutine có thể hoặc không thể là một thủ tục biến cố. Thủ tục không biến cố là thủ tục con đa dụng mà bạn có thể viết và thêm vào bất kỳ chương trình nào.

Ví dụ 15.1 minh họa một thủ tục biến cố trong hầu hết chương trình trong sách.

Ví dụ 15.1. *Thủ tục biến cố chung.*

```
1: Sub cmdExit_Click()
2: End
3: End Sub
```

Khái niệm mới

***Wrapper line** là các dòng mã lệnh bắt đầu và kết thúc thủ tục.*

Dòng 1 và 3, là các dòng bao (wrapper line), xác nhận rằng thủ tục biến cố Click của nút lệnh Exit là một thủ tục con.

Thủ tục biến cố là thủ tục con đặc trưng gắn trực tiếp với các biến cố điều khiển. Thực tế, sẽ có nhiều lần bạn viết thủ tục con không gắn vào bất kỳ biến cố nào. Khi viết một đoạn mã lệnh mà ứng dụng của bạn phải thi hành nhiều lần, thủ tục con không biến cố, đa dụng là ý kiến hay đối với đoạn mã lệnh đó.

Khái niệm mới

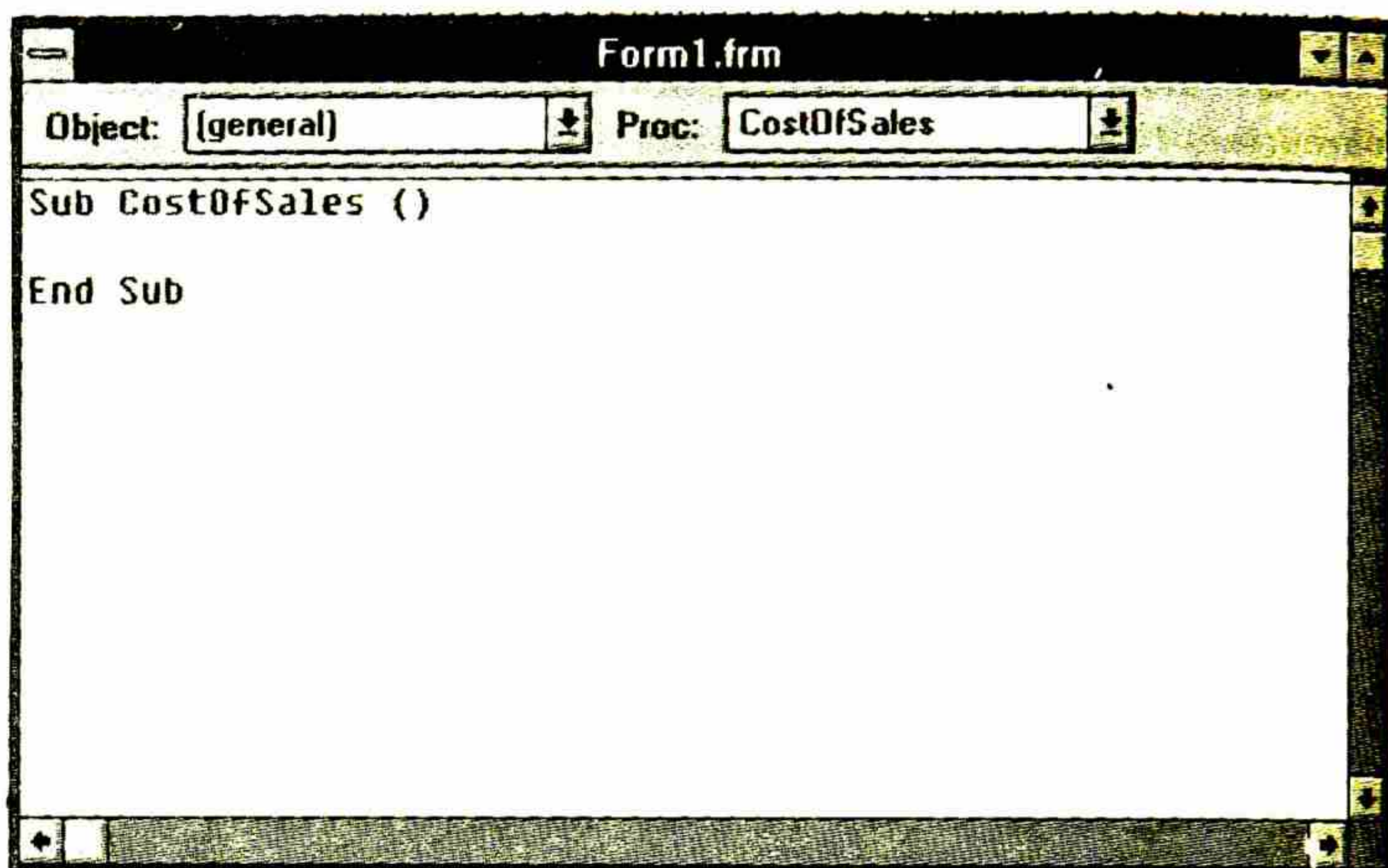
Gọi một thủ tục con nghĩa là *thi hành thủ tục con (subroutine) từ nơi khác trong chương trình.*

Giả sử công ty của bạn có một thủ tục chuyên tính chi phí lương. Trong khi viết một chương trình nhập liệu tính lương, bạn thấy rằng phải thi hành mã lệnh này ở nhiều nơi khác nhau trong chương trình. Nói cách khác, bạn không bao giờ thi hành thủ tục con trên một vòng lặp, thay vào đó bạn có thể tìm ba hay bốn thủ tục biến cố khác nhau cần cho cùng việc tính toán này. Thay vì nhập chính xác mã lệnh vào ba hay bốn nơi khác nhau, hãy nhập mã lệnh chỉ một lần trong thủ tục con của riêng nó và gọi thủ tục con đó từ mỗi thủ tục biến cố cần tính toán.

Hãy thực hiện các bước sau để tạo một thủ tục con không biến cố:

1. Tạo tên cho thủ tục bằng cách dùng qui ước đặt tên bạn đã sử dụng với biến. Chắc chắn tên thủ tục con có một ý nghĩa nào đó. Nếu bạn đang tính chi phí lương, tên CostOfSales sẽ là tên không chê vào đâu được.
2. Mở cửa sổ Code nếu nó chưa mở.
3. Nhấn phím PgDn hay nhấp thanh cuốn dọc của cửa sổ Code để di chuyển cửa sổ Code xuống cuối thủ tục biến cố bất kỳ. Nói cách khác, vị trí con trỏ văn bản của cửa sổ Code ở cuối lệnh End Sub bất kỳ mà bạn có thể tìm thấy.
4. Trên một dòng trống dưới lệnh End Sub, nhập lệnh Sub theo sau là một tên mới của thủ tục con của bạn. Nói cách khác, bạn sẽ nhập **Sub CostOfSales** sau lệnh End Sub của thủ tục biến cố bất kỳ.
5. Ngay khi bạn nhập lệnh Sub, Visual Basic sẽ nhận biết rằng bạn đang bắt đầu một thủ tục con mới. Visual Basic sẽ hoàn thành dòng bao End Sub và hiển thị thủ tục con mới của bạn trong cửa sổ Code, như trong Hình 15.2.

   Nếu đã mở danh sách cuộn xuống Proc, bạn sẽ thấy thủ tục con đa dụng CostOfSales mới trong số các danh sách thủ tục.



Hình 15.2. *Visual Basic chọn thủ tục con của bạn một vị trí riêng trong cửa sổ Code.*

Ghi chú

Lưu ý rằng Visual Basic luôn luôn thêm các dấu ngoặc đơn sau tất cả tên thủ tục con. Nếu bạn không làm gì sẽ giải thoát các dấu ngoặc đơn này.

6. Bây giờ bạn có thể chèn thân thủ tục con vào giữa hai lệnh bao. Khi bạn thêm một dòng và nhấn Enter, Visual Basic sẽ thêm các dòng phụ, cho nên thủ tục con sẽ phát triển khi nó cần. Ví dụ 15.2 là một ví dụ mẫu về thủ tục con tính chi phí lương có thể sử dụng cho thủ tục.

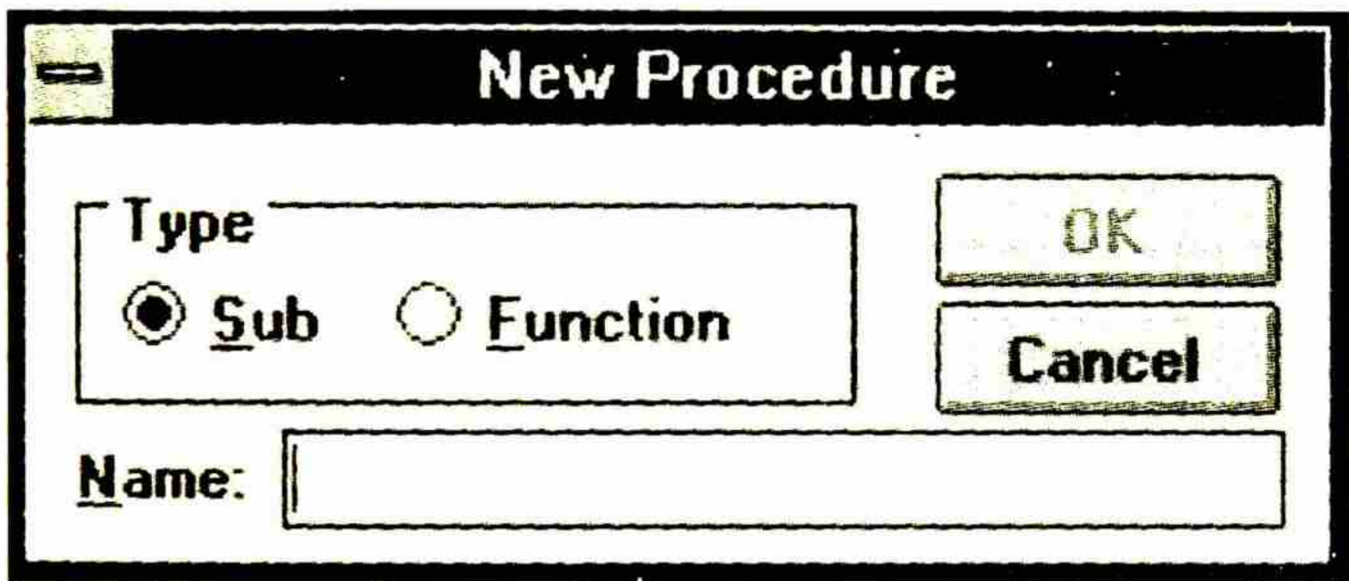
Ví dụ 15.2. *Thủ tục chi phí lương.*

```
Sub CostOfSales ()
' Computes a cost of sales and displays
' that cost in the appropriate label
Dim GrossSales As Currency
Dim CostSales As Currency
Dim OverHead As Single
Dim InventoryFctr As Single
Dim PilferFctr As Single

```

```
' Store initial values from the form
GrossSales = txtGross.Text
InventoryFctr = txtTotalInv.Text * .38
PilferFctr = txtPilfer.Text
OverHead = .21 ' Fixed overhead

CostSales = GrossSales – (InventoryFctr * GrossSales)
CostSales = CostSales – (PilferFctr * GrossSales)
CostSales = CostSales – (OverHead * GrossSales)
lblCost.Caption = Format$(CostSales, "Currency")
End Sub
```



Hình 15.3. *Bạn có thể sử dụng hộp thoại New Procedure để mở một thủ tục mới.*

Thân của thủ tục con tính chi phí lương chỉ dài 18 dòng, nên bạn không muốn nhập 18 dòng đó trong từng thủ tục biến cố cần thi hành. Nếu bạn đã nhập mã lệnh trong thủ tục con riêng của nó như trong Ví dụ 15.2, bạn không bao giờ phải nhập lại mã lệnh nữa. Thay vào đó, bạn chỉ cần gọi thủ tục con đó từ các nơi trong chương trình cần mã lệnh thi hành.

Lời nhắc

Bạn có thể tạo một phím tắt khi muốn mở thủ tục mới. Hãy chọn View New Procedure từ thanh menu. Visual Basic sẽ hiển thị hộp thoại New Procedure nhỏ như trong Hình 15.3. Nếu nhấp Sub sẽ mở một thủ tục Subroutine mới, và nhấp Function sẽ mở một thủ tục Function mới (được mô tả sau trong bài này). Hãy nhập tên thủ tục mới trong hộp nhập Name và nhấp OK. Visual Basic sẽ mở thủ tục mới và nhập các dòng bao đầu cuối cho bạn.

Lệnh Call sẽ thi hành các thủ tục con. Sau đây là dạng thức lệnh Call:

[Call] subName [(argumentList)]

Bài 16 sẽ mô tả công dụng của danh sách đối số *argumentList*. Không phải tất cả thủ tục đều có một danh sách đối số. Lệnh Call là tùy chọn, nhưng nếu bạn sử dụng từ khóa Call khi gọi các thủ tục con, bạn cũng phải bao gồm cả dấu ngoặc đơn. Hãy nhớ: Nếu không có lệnh Call, không có cặp dấu ngoặc đơn; nếu có lệnh Call, sử dụng cặp dấu ngoặc đơn. Nói cách khác, danh sách đối số có thể hoặc không thể được yêu cầu.

Lời nhắc

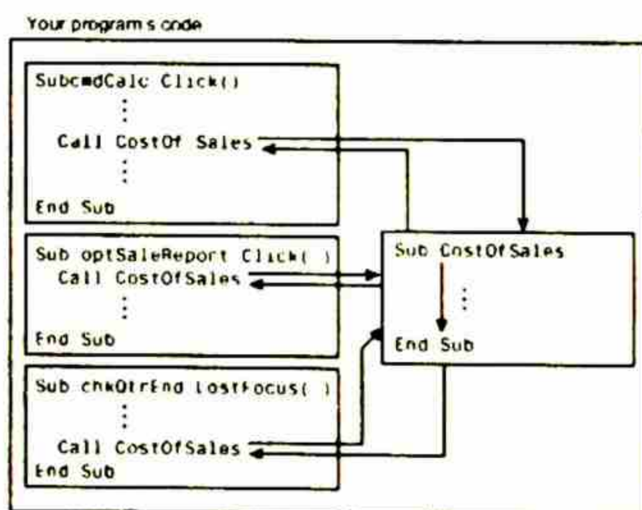
Bạn dễ dàng di chuyển từ thủ tục này sang thủ tục khác trong cửa sổ Code bằng cách dùng các phím PgUp và PgDn. Nếu nhấn F2, phím truy xuất nhanh cho lệnh Window Procedures, Visual Basic sẽ hiển thị danh sách thủ tục; hãy chọn một thủ tục và Visual Basic dẫn bạn đến thủ tục đó trong cửa sổ Code.

Bạn sẽ học về danh sách đối số trong bài tiếp theo. Còn bây giờ, hãy tập trung vào việc sử dụng lệnh Call. Bất cứ khi nào một thủ tục bất kỳ trong chương trình của bạn cần kích hoạt việc thi hành của thủ tục con CostOfSales, bạn chỉ cần mã lệnh cho câu lệnh Call như sau :

```
Call CostOfSales () ' Executes all of the subroutine
```

Câu lệnh sau là tương đương bởi vì không có Call, bạn không cần cặp dấu ngoặc đơn:

```
CostOfSales ' Executes all of the subroutine
```



Hình 15.4. *Các thủ tục Subroutine làm việc bên trong chương trình của bạn.*

Nếu ba thủ tục khác nhau cần thủ tục tính chi phí bán hàng, không gì dễ hơn là nhập mã lệnh một lần trong thủ tục con của riêng nó và sau đó chỉ việc gọi thủ tục con đó từ mỗi nơi trong chương trình khi cần nó. Bạn có thể gọi thủ tục Subroutine từ thủ tục biến cố cũng như từ thủ tục Subroutine không biến cố khác. Hình 15.4 chỉ rõ sơ đồ chương trình vào thời điểm nhiều biến cố và thủ tục Subroutine gọi thủ tục CostOfSales. Khi một trong các thủ tục đang gọi gọi CostOfSales, CostOfSales được nạp và thi hành ngay, việc thi hành chương trình sẽ tiếp tục nơi nó rẽ nhánh trước đó.

Lệnh Call cũng thi hành thủ tục biến cố. Bạn có thể, ngay cả khi người dùng không kích hoạt biến cố, kích một biến cố bất kỳ từ trong mã lệnh bằng cách sử dụng lệnh Call để thi hành thủ tục biến cố. Giả sử nút lệnh Exit thực hiện nhiều nhiệm vụ kết thúc chương trình như xóa mẫu biểu, yêu cầu người dùng cho phép thoát bên trong một hộp thông báo, viết các tập tin dữ liệu cuối cùng lên đĩa, và in báo cáo cuối cùng ra máy in. Nếu bạn hay rằng bạn cần thi hành một chương trình Exit ngay cả khi người dùng chưa nhấp nút lệnh Exit, bạn có thể tự gọi thủ tục biến cố cmdExit_Click().

Ghi chú

Nếu cần thoát một thủ tục Subroutine trước khi thủ tục dừng bình thường – ví dụ, nếu người dùng hủy một hộp nhận dữ liệu trong thủ tục con – hãy sử dụng lệnh Exit Sub.

Ôn lại

Khi các dòng mã lệnh giống nhau xuất hiện nhiều hơn trong chương trình, hãy xét cách đặt các dòng này trong một thủ tục Subroutine riêng. Bạn cần khởi động một thủ tục mới chỉ khi sử dụng Sub hay chọn lệnh New Procedure từ menu View. Định lệnh Call gọi thủ tục Subroutine trong tất cả vị trí chương trình khác mà bạn muốn mã lệnh subroutine thi hành. Bạn có thể gọi thủ tục Subroutine hoặc thủ tục biến cố bằng cách sử dụng Call. Khi Visual Basic thi hành đầy đủ thủ tục, Visual Basic trở lại thi hành mã lệnh đã thi hành lệnh Call.

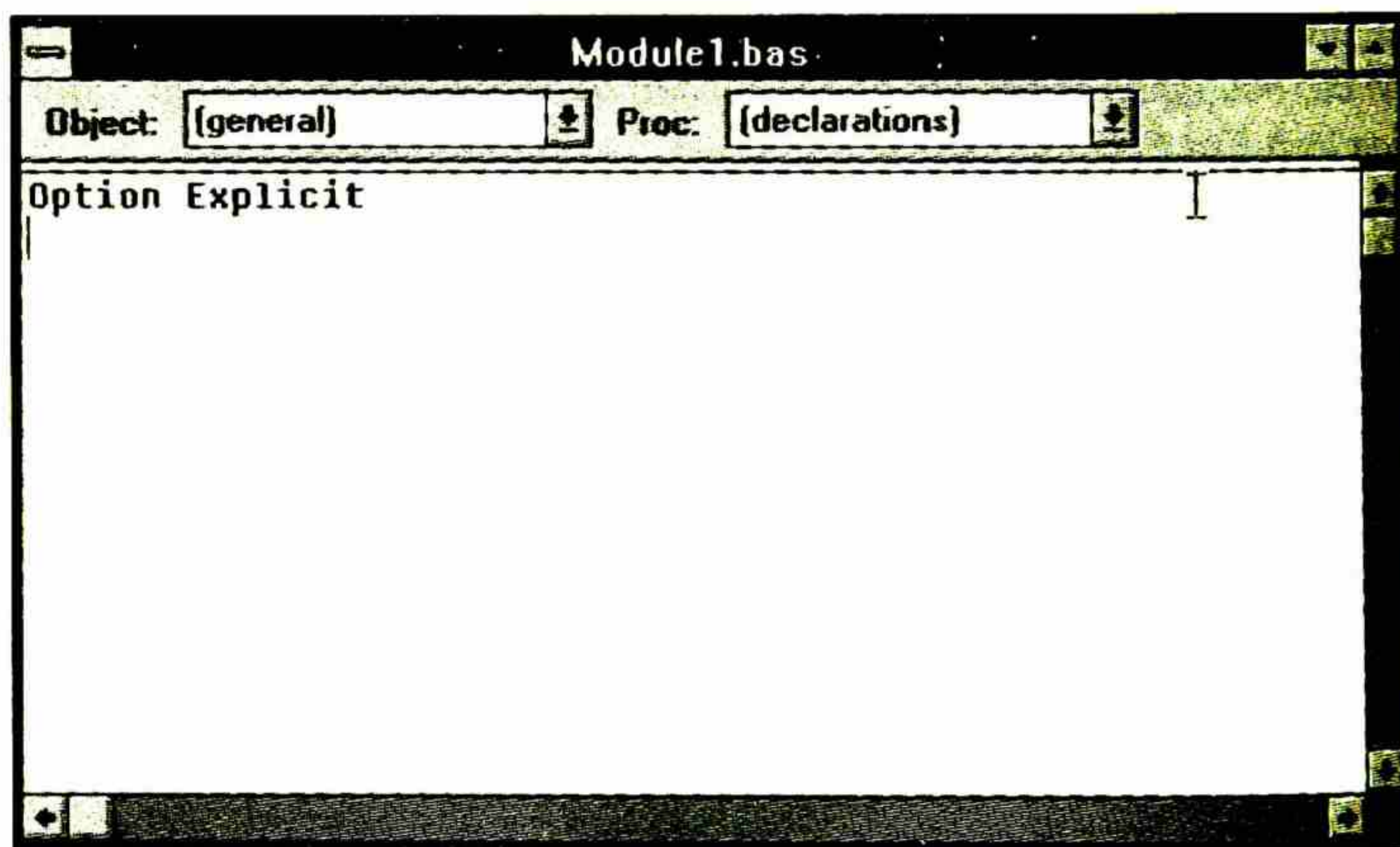
MODULE CÓ PHẦN MỞ RỘNG .BAS

Khái niệm mới

Module *chứa một mẫu biểu và được mẫu biểu sử dụng.*

Mỗi chương trình bạn gặp trước đây đều là module đơn. Module là một tập hợp thủ tục biến cố, thủ tục Subroutine, và thủ tục Function, cùng với một mẫu biểu giữ các điều khiển. Với Visual Basic, bạn có thể thêm các module mã lệnh bổ sung vào module mẫu biểu của bạn. Module mã lệnh là một tập tin bên ngoài, được lưu riêng biệt trên đĩa, chứa mã lệnh các thủ tục bạn viết. Những thủ tục này là thủ tục đa dụng thực hiện việc tính toán dữ liệu. Bạn sẽ học cách chuyển đi và chuyển đến các thủ tục này.

Tập tin CONSTANT.BAS là một loại module đặc biệt chỉ có một thủ tục (General). Mặc dù bạn có thể thêm các thủ tục bổ sung vào CONSTANT.BAS, bạn không nên thêm bớt nội dung trong CONSTANT.BAS. Với từng phiên bản Visual Basic do Microsoft phát hành, Microsoft sẽ cập nhật nội dung CONSTANT.BAS, và bạn muốn thay thế tập tin CONSTANT.BAS cũ bằng một tập tin mới khi nâng cấp lên một phiên bản Visual Basic khác.



Hình 15.5. *Cửa sổ Code mới cho tập tin module bên ngoài MODULE1.BAS.*

Bạn có thể thêm các module mã lệnh vào chương trình bằng cách chọn File New Module (hay nhấp nút New Module trên thanh công cụ, nút thứ hai kể từ trái sang). Ngay khi bạn yêu cầu một module mới, Visual Basic sẽ mở một cửa sổ Code mới cho module đó. Visual Basic tự động đặt tên module đầu tiên mà bạn thêm là MODULE1.BAS (bổ sung tập tin đó vào danh sách tập tin của cửa sổ Project cho ứng dụng đó). Nếu bạn đã thêm một module thứ hai, Visual Basic sẽ đặt tên module đó là MODULE2.BAS, và cứ tiếp tục. Hình 15.5 cho thấy cửa sổ Code mới mở khi bạn thêm một module mới vào ứng dụng.

Một khi thêm mã lệnh vào tập tin module, bạn nên cất tập tin dưới một tên khác với tên mặc định. Tuy nhiên, vẫn duy trì phần mở rộng tập tin .BAS. Trong Visual Basic phần mở rộng tên tập tin .BAS thường dành cho các tập tin module bên ngoài đa dụng mà bạn sẽ viết và sử dụng.

Cửa sổ Project cho từng ứng dụng sẽ làm mới tập tin mẫu biểu và tập tin module (cũng như tập tin CONSTANT.BAS nếu bạn sử dụng nó) mà mỗi ứng dụng yêu cầu. Khi tạo module, bạn đang tạo một tập tin độc lập trên đĩa cho nhiều ứng dụng có thể sử dụng ngay khi bạn thêm module đó vào cửa sổ Project của ứng dụng.

Giả sử bạn đã viết nhiều thủ tục Subroutine in tên công ty và địa chỉ của bạn, tính thuế doanh thu cho việc bán hàng trong 20 vùng thuộc bộ phận của bạn, và thực hiện việc lưu trữ các tập tin dữ liệu nếu dữ liệu thuộc tháng đầu tiên. Có lẽ bạn nhận thấy rằng, mặc dù bạn đã viết các thủ tục cơ bản này cho một hệ thống sổ cái, bạn vẫn cần những thủ tục này trong các thủ tục chương trình khác. Bạn có thể mở một module mới từ trong cửa sổ Code của ứng dụng sổ cái, và cắt dán các thủ tục này từ ứng dụng sổ cái vào module mới. Sau đó nếu bạn cất module dưới tên MYPROC.BAS, ứng dụng sổ cái sẽ có thể gọi các thủ tục của module đó (thực tế là Visual Basic đã thêm module vào cửa sổ Project khi bạn tạo module lần đầu). Cũng như bất kỳ một ứng dụng khác, bạn có thể chọn *File Add* và thêm module *MYPROC.BAS* vào.

Tất cả chương trình tự động có một module mẫu biểu. Module bạn làm việc, bằng cách sử dụng cửa sổ Code, luôn hiển thị khi bạn mở một thủ tục biến cố của mẫu biểu hay của điều khiển, là module mẫu biểu. Module bổ sung bạn thêm vào ứng dụng, cũng như module được hỗ trợ trong CONSTANT.BAS, đôi khi được gọi là module không mẫu biểu.

Ôn lại

Bạn thường tạo các tập tin module khi chạy qua những thủ tục có ích. Bạn có thể sở hữu nhiều module trên đĩa như bạn muốn và thêm bất kỳ hay tất cả chúng vào những ứng dụng bạn viết sau này. Với cách này, bạn sẽ xây dựng các ứng dụng mới sau đó nhanh hơn bằng cách sử dụng lại mã lệnh (qua lệnh Call) mà bạn đã viết cho các ứng dụng khác và được lưu trong những module đa dụng.

THỦ TỤC FUNCTION

Khái niệm

Như với hàm xây dựng sẵn, bạn có thể viết các thủ tục Function đa dụng riêng của bạn, thường gọi là *function*, mà không gắn với các biến cố xác định. Bạn có thể gọi hàm này từ ứng dụng Visual Basic, đặt hàm này trong module mã lệnh bên ngoài riêng biệt và thêm chúng vào cửa sổ Code của ứng dụng. Thủ tục Function làm việc giống như thủ tục Subroutine; bạn có thể gọi chúng từ bất kỳ nơi nào khác trong chương trình. Tuy nhiên, không giống thủ tục Subroutine, thủ tục Function trả về các giá trị.

Bạn có thể viết thủ tục Function riêng để tăng thêm số thủ tục xây dựng sẵn được Visual Basic hỗ trợ. Nếu thi hành một phép biến đổi đơn vị đo, bạn sẽ thấy rằng bạn cần công việc chuyển đổi này ở nhiều nơi trong chương trình. Vì thế hãy thêm mã lệnh đó vào một thủ tục Function. Hàm sẽ thực hiện việc tính toán và trả về câu trả lời cho thủ tục con đang gọi.

Khi gọi các hàm xây dựng sẵn, bạn phải thực hiện điều gì đó với giá trị trả về. Bạn không thể viết hàm Int() đứng một mình trên một dòng như sau:

```
Int(Amount) ' Invalid!
```

Int(Amount) sẽ đổi số amount thành một số nguyên đồng thời trả về số nguyên đó, và bạn phải thực hiện một điều gì đó với số nguyên được trả về. Vì vậy, thường thường bạn sẽ gán giá trị trả về như sau:

```
lblAmt.Caption = Int(Amount)
```

Hay bạn sử dụng hàm cho việc tính toán cần giá trị này:

```
WholePart = Int(Amount) + Estimate
```

Bạn có thể viết mã lệnh cho các thủ tục Function số và chuỗi riêng cũng như thêm chúng vào tập hợp hàm của Visual Basic. Những hàm bạn viết không hoàn toàn là hàm xây dựng sẵn của Visual Basic bởi vì các hàm của bạn không là một phần của ngôn ngữ Visual Basic. Tuy nhiên, như với thủ tục. Subroutine, bạn có thể viết các thủ tục Function trong cửa sổ Code của ứng dụng cũng như lưu thủ tục Function trong cửa sổ Code hay cùng với những thủ tục Subroutine trong các module bên ngoài để nhiều ứng dụng có thể truy xuất tới hàm.

Để viết thủ tục Function mới trong cửa sổ Code, dù cửa sổ Code là cửa sổ Code của mẫu biểu trong ứng dụng hay module bên ngoài .BAS mà bạn đã mở, bạn có thể chọn View ➡ New Procedure từ menu và nhấp nút chọn lựa Function trên hộp thoại New Procedure. (Trong phần trước, bạn đã nhấp Sub để mở một thủ tục con mới.) Sau khi bạn nhập một tên mới cho hàm trong hộp nhập Name, Visual Basic sẽ thêm các dòng bao đầu cuối hàm giống như ví dụ dưới đây cho một thủ tục Function mới tên là MultiplyIt:

```
Function MultiplyIt ()
End Function
```

Thủ tục Function luôn bắt đầu bằng lệnh Function và kết thúc với lệnh End Function. Bạn sẽ thêm mã lệnh tới thân hàm giữa hai dòng bao.

Lời nhắc

Nếu bạn thêm dấu $ sau tên hàm, chẳng hạn như Reverse$(), Visual Basic cho rằng bạn muốn mở một hàm chuỗi sẽ trả về một giá trị chuỗi. Còn như bạn bỏ qua dấu $, Visual Basic cho rằng hàm của bạn sẽ trả về kiểu dữ liệu Variant mà trong đó bạn có thể trả về giá trị số hay chuỗi. Để mọi việc rõ ràng, luôn sử dụng dấu $ cho hàm chuỗi.

Muốn trả về giá trị hàm, hãy gán giá trị trả về cho tên hàm. Đừng sử dụng Call để gọi hàm. Tất cả những gì bạn làm để gọi một hàm là sử dụng tên thủ tục Function trong biểu thức hay trong lệnh.

Ghi chú

Trường hợp bạn cần thoát một thủ tục Function trước khi các hàm dừng thông thường, hãy sử dụng lệnh Exit Function.

Tóm tắt

Ví dụ 15.3 chứa một hàm tính bưu phí cho thư từ hay hàng hóa như sau:

1. Bưu điện sẽ tính 32 cent cho 8 ounce đầu tiên.
2. Thêm 15 cent cho từng 4 ounce tiếp theo.
3. Trọng lượng phải dưới 24 ounce.

Ví dụ 15.3 giả sử rằng trọng lượng thư hay gói hàng sẽ xuất hiện trong một điều khiển hộp nhập (Text Box) có tên là txtWeight.Text. Thêm vào đó, trọng lượng phải tính theo đơn vị ounce. Vì vậy, một ứng dụng bất kỳ sử dụng hàm này phải chắc chắn rằng một hộp nhập có tên txtWeight đang tồn tại và giữ tổng trọng lượng của gói hàng trước khi gọi hàm.

Ôn lại

Khi bạn muốn thủ tục con đặc biệt tính hay tổ hợp giá trị chuỗi, và Visual Basic không hỗ trợ thủ tục này với các hàm xây dựng sẵn như CInt() và Mid$(), hãy viết thủ tục Function cho riêng bạn. Trước khi thủ tục Function dừng, gán giá trị trả về của hàm cho tên hàm.

Ví dụ 15.3. *Một hàm tính tiền gửi bưu kiện và trả về số tiền.*

```
Function Postage ()
' Calculate postage based on the
' weight of a letter or package
Dim PostHold As Currency
Dim weight As Integer

' Grab the weight value from the text box
' and convert to number for comparison
weight = Val(txtWeight.Text)

Select Case weight
Case Is <= 8:
PostHold = .32
Case Is <= 12:
PostHold = .47
```

```
16: Case Is <= 16:
17: PostHold = .62
18: Case Is <= 20:
19: PostHold = .77
20: Case Is < 24:
21: PostHold = .92
22: Case Is >= 24:
23: MsgBox "Weight is more than 24 ounces!",
    MB_ICONEXCLAMATION, "Error"
24: PostHold = 0
25: End Select
26:
27: Postage = PostHold ' Return the value
28: End Function
```

Phân tích

Dòng 3 định nghĩa một biến nắm giữ số tiền chuyển bưu phí qua lệnh Select Case. Thực tế, biến PostHold không hoàn toàn cần thiết bởi vì mỗi tùy chọn Case có thể gán trực tiếp tới tên hàm. Tuy nhiên, việc gán cho tên hàm ở cuối hàm sẽ làm mã lệnh rõ ràng hơn. Khi hàm hoàn thành, dòng 26 hoàn tất nhiệm vụ của hàm bằng cách gán giá trị được tính toán cho tên hàm.

Bài tiếp theo sẽ chỉ cho bạn cách chia sẻ các biến giữa các hàm. Những điều khiển bạn đang dùng có hiệu lực với tất cả thủ tục trong chương trình nhưng không là biến. Chỉ khi bạn học cách chuyển dữ liệu từ một thủ tục sang thủ tục khác, bạn mới có thể thấu hiểu sức mạnh thực sự của thủ tục Function và Subroutine.

Bài tập

Kiến thức tổng quát

1. Đúng hay Sai: Thủ tục Subroutine giống thủ tục biến cố.
2. Bạn có thể gọi thủ tục Subroutine như thế nào?
3. Bạn có thể gọi thủ tục Function như thế nào?

4. Nêu hai cách nhập một thủ tục mới trong cửa sổ Code.
5. Bằng cách nào bạn có thể xác định một thủ tục Function trả về kiểu dữ liệu chuỗi hay Variant?
6. Điểm khác biệt chính giữa thủ tục Subroutine và thủ tục Function?
7. Bạn có thể cho biết giá trị trả về của một hàm như thế nào?
8. Phím tắt di chuyển từ thủ tục này sang thủ tục khác là gì?
9. Một thủ tục Function do lập trình viên định nghĩa khác với một hàm có sẵn như thế nào?
10. Tên tập tin chứa các thủ tục (General) là gì?
11. Các tập tin module có phần mở rộng là gì?
12. Tên hai câu lệnh có thể xuất hiện trong thủ tục (General).
13. Đúng hay Sai: Bạn có thể gọi các thủ tục biến cố từ những thủ tục biến cố khác.
14. Đúng hay Sai: Bạn có thể gọi thủ tục biến cố từ thủ tục Subroutine.

Viết mã lệnh...

15. Viết hai dòng lệnh khung cho một thủ tục Subroutine tên PrReport().
16. Viết hai dòng lệnh khung cho một thủ tục Function tên GetValue().
17. Giả sử bạn đã viết một hàm tên GetPi() trả về giá trị *pi* (xấp xỉ 3.14159). Hãy viết lệnh lấy giá trị trả về đó.

Tìm lỗi kỹ thuật

18. Cho biết có gì không ổn với lệnh Call này:

```
Call MySub arg1, arg2
```

Phần nâng cao

Hãy viết một hàm đảo chuỗi tên Reverse$() lấy chuỗi lưu trong hộp nhập txtAString và đảo chuỗi, trả về chuỗi trong tên hàm Reverse.

Bài 16

Đối số và phạm vi

- **Ba loại phạm vi biến**
- **Biến toàn cục**
- **Biến module**
- **Biến cục bộ – biến an toàn nhất**
- **Chuyển đối số**
- **Tham biến và tham trị**

Bây giờ bạn có thể viết các chương trình con – bao gồm thủ tục Subroutine và thủ tục Function được lưu trong module – Lúc này là thời điểm thích hợp để học cách các thủ tục này liên lạc với nhau. Cho đến bây giờ, những biến bạn đã định nghĩa gọi là *biến cục bộ* (local variable). Biến cục bộ chỉ được biết đến trong thủ tục bạn định nghĩa biến.

Nếu hai hay nhiều thủ tục phải làm việc với cùng một giá trị, giá trị đó phải là giá trị từ một điều khiển trên mẫu biểu. Các điều khiển có hiệu lực với hết thảy thủ tục trong ứng dụng. Tuy nhiên, không phải khi nào điều khiển cũng dành cho tất cả giá trị, đặc biệt giá trị trung gian cần trong nhiều ứng dụng lớn. Chỉ những giá trị hiển thị tới người dùng mới được lưu trong các điều khiển. Những giá trị trung gian còn lại phải ở trong biến và có tác dụng với tất cả thủ tục làm việc với biến đó.

BA LOẠI PHẠM VI BIẾN

Khái niệm mới

Phạm vi (scope) xác định số lượng chương trình có thể truy xuất một biến.

Bạn đã biết những kiểu dữ liệu mà biến có thể lưu giữ. Đó là biến nguyên, chuỗi, thập phân, cũng như biến Variant có thể chấp nhận một kiểu dữ liệu bất kỳ. Biến không chỉ có tên, nội dung và kiểu dữ liệu, mà còn phải thuộc một trong ba loại phạm vi sau:

- Phạm vi biến cục bộ
- Phạm vi biến module
- Phạm vi biến toàn cục

Ghi chú

Nếu đã bắt gặp CONSTANT.BAS, tập tin module bạn thêm vào AUTOLOAD.VBP, bạn sẽ thấy tất cả giá trị có tên của nó được định nghĩa bên trong thủ tục (General). Vị trí những định nghĩa này và định nghĩa sử dụng lệnh Global *sẽ cho biết hết thảy giá trị bên trong CONSTANT.BAS được định nghĩa ở mức module. Nếu Microsoft đã định nghĩa tất cả giá trị tên hằng trong CONSTANT.BAS từ một thủ tục CONSTANT.BAS hơn là thủ tục (General), thì không có thủ tục nào khác có thể truy xuất các giá trị trong CONSTANT.BAS.*

Khi cần định nghĩa biến, bạn nên sử dụng lệnh Dim. Bây giờ bạn đã biết cách viết chương trình có nhiều thủ tục; một hay nhiều thủ tục có lẽ cần chia sẻ dữ liệu. Bạn phải quyết định không chỉ cách định nghĩa biến (sử dụng Dim hay Global), mà còn nơi bạn muốn định nghĩa những biến này. Việc lựa chọn phụ thuộc vào phạm vi biến. Nếu một biến chỉ dùng trong thủ tục đơn, không có lý do gì để định nghĩa biến đó có phạm vi module hay toàn cục.

Ôn lại

Phạm vi dữ liệu sẽ xác định phần mở rộng khác của chương trình có thể truy xuất dữ liệu đó. Các điều khiển luôn luôn có phạm vi toàn cục bởi vì thủ tục bất kỳ trong một module chương trình nào đều có thể sử dụng dữ liệu điều khiển. Các biến đã định nghĩa trong thủ tục (General) của một module sử dụng từ khóa Global cũng là toàn cục và có hiệu lực trong chương trình. Các biến đã định nghĩa trong thủ tục (General) sử dụng lệnh Dim là những biến module và có hiệu lực cho bất kỳ thủ tục nào trong module. Các biến đã định nghĩa bằng cách sử dụng Dim trong các thủ tục (không phải thủ tục (General)) là biến cục bộ và có hiệu lực chỉ trong thủ tục đó.

BIẾN TOÀN CỤC

Khái niệm

Lệnh Global sẽ định nghĩa biến toàn cục. Bạn có thể sử dụng Global chỉ trong một thủ tục (General) của module. Thông thường, các lập trình viên thêm từ khóa Const, như đã làm trong CONSTANT.BAS, để định nghĩa tên các giá trị hằng mà giá trị của chúng không thể thay đổi.

Lệnh Global tương tự lệnh Dim. Sau đây là dạng thức lệnh Global:

Global [Const] *VarNa me* [AS *DataType*] [= *value*]

Bạn có thể định nghĩa biến là biến toàn cục chỉ trong thủ tục khai báo của một module. Từng thủ tục trong toàn bộ ứng dụng đó có thể truy xuất các biến toàn cục. Vì vậy, thủ tục Form_Load() của một module có thể khởi tạo biến toàn cục mà bạn đã định nghĩa, và thủ tục Function của module khác có thể truy xuất và thay đổi biến đó. Nếu sử dụng từ khóa Const, bạn cũng phải gán cho biến một giá trị ban đầu nhưng không sử dụng kiểu dữ liệu. Không có Const, bạn phải xác định kiểu dữ liệu.

Lời nhắc

Hãy định nghĩa tên hằng bằng chữ hoa. Xuyên suốt chương trình, bạn sẽ có khả năng phân biệt biến với tên hằng.

Lệnh sau xuất hiện trong thủ tục (General) của một module bất kỳ không có mẫu biểu. Hãy định nghĩa biến toàn cục có tên là MyGlobal. Một thủ tục bất kỳ trong toàn bộ ứng dụng có thể khởi tạo, truy xuất, và thay đổi biến.

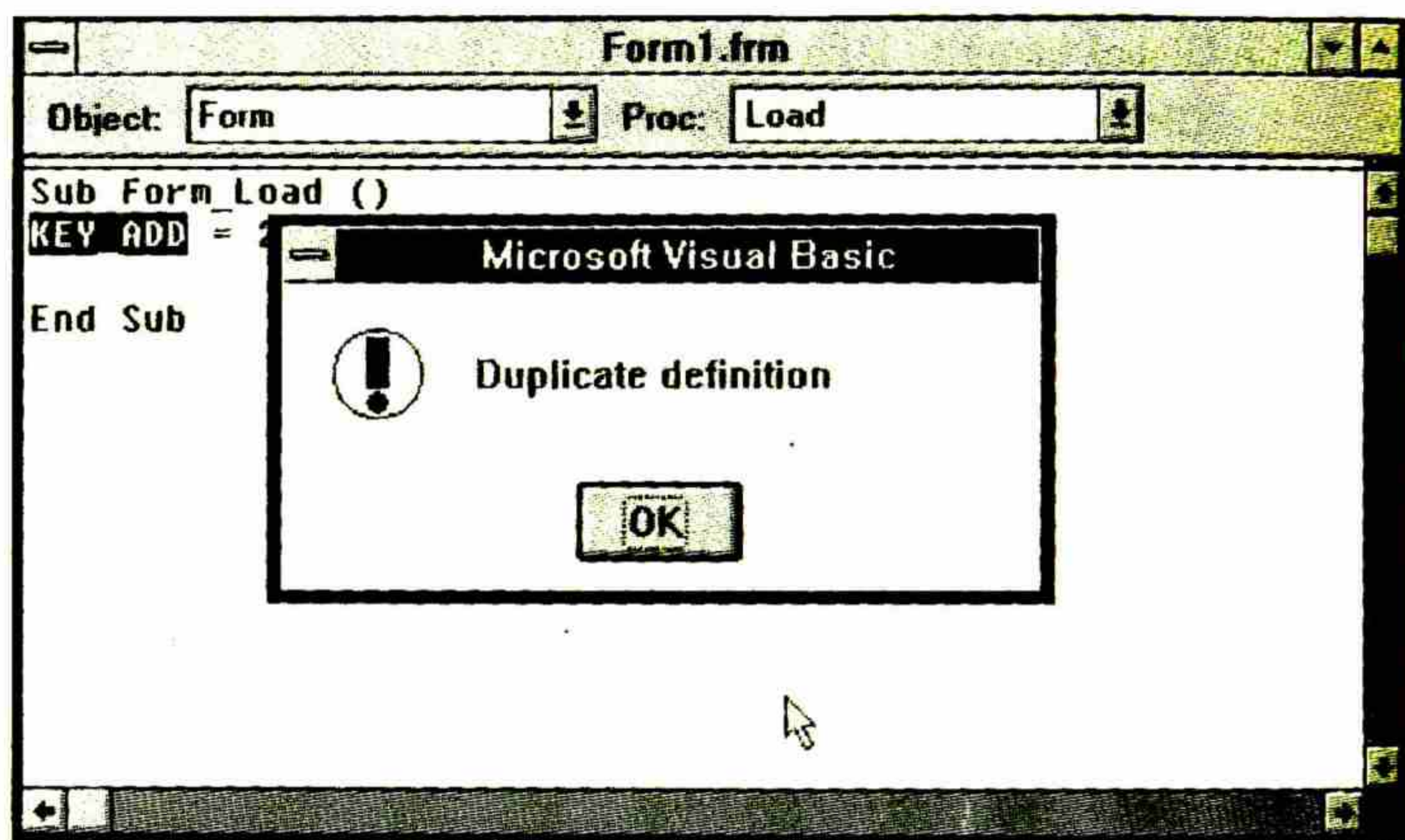
```
Global MyGlobal As Integer
```

Từ khóa Const cho phép bạn đặt tên hằng số sẽ sử dụng trong suốt phần còn lại của chương trình. Ví dụ, nếu số bộ phận trong công ty của bạn là 8, có lẽ bạn muốn định nghĩa một tên hằng toàn cục như sau:

```
Global Const NUMDIVS = 8
```

Có thể không bao giờ thay đổi được tên hằng do người dùng nhập vào, do một phép gán, hay qua bất kỳ một loại biên nào khác thay đổi lệnh ở phần còn lại của chương trình. NUMDIVS sẽ luôn luôn giữ giá trị bằng 8. Thuận lợi của việc sử dụng tên hằng thay vì dùng giá trị 8 ở

mọi nơi trong chương trình cần sử dụng số bộ phận, ở chỗ nếu số bộ phận thay đổi, bạn phải thay đổi số ở một nơi, và phần còn lại của chương trình sẽ tự động sử dụng giá trị mới. Cũng như vậy, chỉ định từ Const giúp bạn hay một ai đó sửa đổi mã lệnh ghi đè giá trị ngẫu nhiên. Hình 16.1 minh họa hộp thông báo lỗi mà Visual Basic sẽ hiển thị nếu một dòng bất kỳ trong chương trình cố thay đổi tên hằng.



Hình 16.1. *Visual Basic sẽ nhắc nhở bạn nếu cố thay đổi một hằng đã đặt tên.*

Lý do bạn phải xác định giá trị hằng ban đầu là vì lệnh Global chỉ đặt trong một chương trình nơi bạn có thể gán giá trị cho hằng.

Tóm tắt

Ví dụ 16.1 có ba biến toàn cục và hai định nghĩa tên hằng toàn cục. Mã lệnh phải xuất hiện trong thủ tục (General) của một module không mẫu biểu bằng không Visual Basic sẽ đưa ra thông báo lỗi.

Ôn lại

Lệnh Global cho phép bạn định nghĩa biến toàn cục. Nếu sử dụng bổ từ Const để đặt tên cho các hằng toàn cục, bạn không thể định nghĩa kiểu dữ liệu toàn cục, nhưng bạn phải khởi tạo hằng toàn cục tại thời điểm định nghĩa tên hằng.

Ví dụ 16.1. *Định nghĩa năm giá trị toàn cục.*

```
1: Option Explicit
2:
3: ' Three global variables
4: Global G1 As Single
5: Global G2 As Double
6: Global G3 As Single
7:
8: ' Two named constants
9: Global Const G4 = 495.42
10: Global Const G5 = 0
```

Phân tích

Dòng 1 xác định rằng tất cả biến trong module phải được định nghĩa rõ ràng. Lệnh Option Explicit có nghĩa là không cho một thay đổi nhỏ nào do lỗi đánh vần tên biến. Dòng 4 đến 6 định nghĩa ba biến toàn cục. Một thủ tục bất kỳ trong toàn bộ chương trình, ngay cả thủ tục trong các module khác, có thể khởi tạo, đọc và thay đổi G1, G2, và G3. Các dòng 9 và 10 sẽ định nghĩa hai giá trị tên hằng toàn cục gọi là G4 và G5. Cả hai hằng được định nghĩa trong các dòng 9 và 10 bởi vì G4 và G5 không thể được khởi tạo ở bất kỳ nơi nào khác trong chương trình.

BIẾN MODULE

Khái niệm

Biến module cũng được định nghĩa trong thủ tục (General). Ít dùng hơn Global, bạn sử dụng Dim để định nghĩa biến module. Biến module có hiệu lực trong mọi thủ tục trong module được định nghĩa. Vì vậy, biến module có phạm vi giới hạn hơn biến toàn cục bởi vì chúng không có hiệu lực toàn cục.

Biến module được cho là ít an toàn hơn biến toàn cục. Chỉ module trong đó bạn định nghĩa biến module mới có thể truy xuất và thay đổi các biến này. Vì vậy, về mặt kỹ thuật, bạn có thể có hai module khác nhau trong một ứng dụng đơn, và mỗi module có thể có cùng biến module cùng tên. Tuy nhiên, mỗi biến module của module, sẽ phân biệt các biến. Mã lệnh truy xuất và thay đổi biến module trong một module chỉ thay đổi giá trị của biến module đó.

Chú ý

Mặc dù trên thực tế hai module trong cùng ứng dụng có thể đặt tên cùng tên biến mà không có xung đột, song bạn nên cố gắng không đặt tên biến trùng nhau vì rắc rối trong việc bảo trì do các biến định nghĩa trùng tên gây nên.

Không giống biến toàn cục, có thể định nghĩa chỉ trong các module không mẫu biểu, bạn dễ dàng định nghĩa biến module trong module của mẫu biểu hay trong một module bên ngoài bất kỳ thuộc chương trình của bạn.

Tóm tắt

Ví dụ 16.2 sẽ định nghĩa hai biến module. Các biến module phải được định nghĩa trong thủ tục (General) của một module bất kỳ thuộc ứng dụng.

Ôn lại

Các biến module, được định nghĩa với Dim trong thủ tục (General) của một module bất kỳ, có hiệu lực tới một thủ tục bất kỳ trong module đó.

Ví dụ 16.2. *Mã lệnh định nghĩa hai biến module.*

```
1: Dim M1 As Integer
2: Dim M2 As Double
```

Phân tích

Dòng 1 và 2 định nghĩa hai biến module tên M1 và M2. Các định nghĩa phải xuất hiện trong cùng thủ tục (General) của module. Biến không có tính toàn cục bởi vì lệnh Dim định nghĩa chúng thay vì lệnh Global.

BIẾN CỤC BỘ – BIẾN AN TOÀN NHẤT

Khái niệm

Biến cục bộ chỉ có hiệu lực trong những thủ tục mà chúng được định nghĩa. Hãy sử dụng lệnh Dim để định nghĩa biến cục bộ. Bạn có thể định nghĩa biến cục bộ chỉ trong thủ tục biến cục bộ chứ không phải thủ tục khai báo riêng biệt; bạn không thể định nghĩa biến cục bộ trong thủ tục (General).

Biến Global và biến module không được xem là an toàn như các biến cục bộ. Giả sử chỉ ba thủ tục trong một ứng dụng lớn cần truy xuất tới một biến. Nếu giới hạn phạm vi của biến tới ba thủ tục này, bạn sẽ không thay đổi tình cờ những biến này trong các thủ tục khác bằng việc sử dụng biến ở nơi bạn không bao giờ định sử dụng chúng.

Biến làm việc trong chương trình của bạn nên là biến cục bộ. Những biến này không có hiệu lực ở ngoài thủ tục của chúng. Tất cả lệnh Dim bạn gặp trong sách đã định nghĩa biến cục bộ.

Chú ý

Biến cục bộ được định nghĩa bằng lệnh Dim sẽ kết thúc ngay khi việc thi hành thủ tục kết thúc.

Khi một thủ tục kết thúc, các biến cục bộ thuộc thủ tục do lệnh Dim định nghĩa sẽ được giải phóng và những giá trị biến mất. Nếu gọi trở lại thủ tục đó, Visual Basic sẽ tái định nghĩa tất cả biến này và khởi tạo chúng một lần nữa. Vì vậy, nếu tiến hành nhập vào một thủ tục tính theo giây, các giá trị của tất cả biến do lệnh Dim định nghĩa vào lần cuối cùng thủ tục được thi hành sẽ không có tác dụng lâu hơn.

Tóm tắt

Ví dụ 16.3 chỉ ra một thủ tục biến cố của nút lệnh sẽ định nghĩa một biến cục bộ và sử dụng biến đó và một biến module để tính giá trị cho đề mục nhãn.

Ôn lại

Khi khai báo biến ở đầu một thủ tục bằng việc sử dụng lệnh Dim, bạn sẽ khai báo biến cục bộ mà Visual Basic chỉ nhận biết trong thời gian thủ tục đó thi hành.

Ví dụ 16.3. *Sử dụng biến module và biến cục bộ trong một thủ tục.*

```
1: Sub cmdInvFactor_Click()
2: ' Adds an inventory factor to a module-level
3: ' inventory total when the user clicks the
4: ' inventory command button
5: Dim FactAdd As Single
6: FactAdd = .13
```

```
7: ' Add to the module variable named InventoryTotal
8: lblInvent.Caption = InventoryTotal + FactAdd
9: End Sub
```

Phân tích

Dòng 1 bắt đầu một thủ tục biến cố mới. Vì vậy, một biến bất kỳ đã được định nghĩa trong thủ tục, bằng cách sử dụng lệnh Dim, phải là biến cục bộ. Dòng 5 sẽ định nghĩa biến cục bộ FactAdd. Dòng 8 sau đó thêm FactAdd vào biến module InventoryTotal để tính kết quả của nhãn cuối cùng.

CHUYỂN ĐỐI SỐ

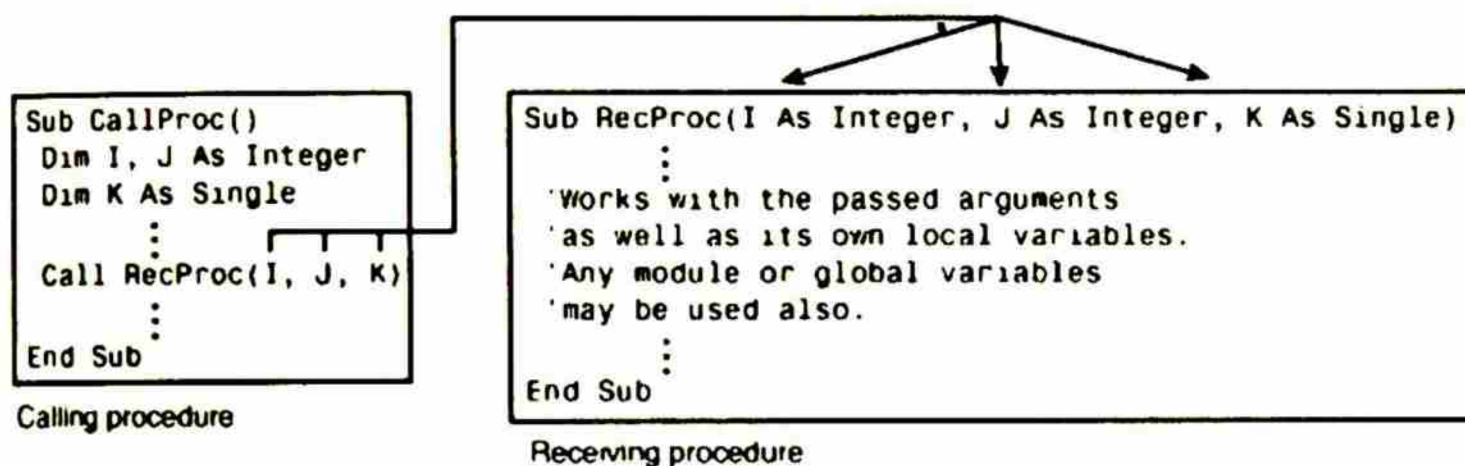
Khái niệm

Biến cục bộ được xem là biến an toàn nhất, là loại biến tốt nhất để định nghĩa. Tuy nhiên, nhiều khi một thủ tục Subroutine hay Function lại cần một giá trị từ biến cục bộ khác. Ví dụ, giả sử một thủ tục tính một giá trị, và thủ tục thứ hai phải sử dụng giá trị đó trong một tính toán khác trước khi hiển thị kết quả trên mẫu biểu. Phần này sẽ giải thích cách chuyển dữ liệu cục bộ từ thủ tục định nghĩa biến cục bộ tới thủ tục khác cần làm việc với giá trị đó.

Khi gọi các hàm xây dựng sẵn, bạn chuyển một hay nhiều đối số tới hàm để mã lệnh bên trong hàm có dữ liệu làm việc. Khi gọi các thủ tục Subroutine và Function của riêng bạn, bạn cũng có thể chuyển các đối số tới chúng. Đối số thường không nhiều hơn biến cục bộ của thủ tục đang chuyển.

Một khi bạn chuyển dữ liệu, dữ liệu vẫn là cục bộ đối với thủ tục chuyển ban đầu, nhưng thủ tục có cơ hội làm việc với các giá trị này. Phụ thuộc vào cách bạn chuyển đối số, thủ tục nhận thậm chí có thể thay đổi các giá trị này để khi thủ tục chuyển chiếm lại quyền điều khiển, các biến cục bộ của nó đã được sửa đổi.

Hình 16.2 minh họa thuật ngữ được sử dụng trong các đối số nhận và chuyển từ một thủ tục tới một thủ tục khác. Thủ tục thường gọi là *thủ tục gọi* (calling procedure), sẽ chuyển một hay nhiều biến cục bộ của nó, là *đối số* (argument), tới *thủ tục nhận* (receiving procedure). Lý do duy nhất các cặp ngoặc đơn tồn tại sau một tên thủ tục là để giữ các đối số được gửi tới hàm nhận.



Hình 16.2. *Hiển thị thuật ngữ gọi thủ tục chính.*

Ghi chú

Như bạn đã biết, nếu thủ tục nhận là một thủ tục Function, thủ tục Function thường trả về một giá trị cho thủ tục gọi.

Trong Hình 16.2, các đối số nhận là I, J, và K. Thủ tục nhận sử dụng cùng tên thủ tục gọi trong hình này, và đó là trường hợp thường gặp. Tuy nhiên, bạn không phải sử dụng cùng tên. Nói một cách khác, nếu RecProc() đã nhận ba biến X, Y, và Z, sau đó RecProc() sẽ có ba biến để làm việc với tên là X, Y, và Z, và ba biến này có cùng giá trị như I, J, và K trong thủ tục gọi. Vì vậy, tên biến nhận không phải đúng với tên chuyển, mặc dù điều đó hạn chế được tình trạng lộn xộn.

Hãy khai báo kiểu dữ liệu của tất cả đối số nhận. Nếu bạn phải chuyển và nhận nhiều đối số, hãy phân cách các đối số được chuyển và nhận (cùng với kiểu dữ liệu khai báo của chúng), với dấu phẩy. Câu lệnh sau sẽ chuyển ba giá trị cho thủ tục con trong Hình 16.2:

```
Call RecProc(I, J, K)
```

Câu lệnh sau bắt đầu thủ tục RecProc():

```
Sub RecProc (I As Integer, J As Integer, K As Single)
```

Thủ tục gọi đã biết các kiểu dữ liệu I, J, và K, nhưng các giá trị này không biết RecProc(). Vì vậy, bạn sẽ phải viết mã lệnh kiểu dữ liệu cho từng đối số được nhận để hàm nhận biết kiểu dữ liệu của các đối số được gửi.

Nếu một thủ tục Subroutine hay Function nhận các mảng, đừng chỉ các chỉ số mảng bên trong danh sách đối số. Lệnh Sub sau sẽ định nghĩa một thủ tục Subroutine đa dụng chấp nhận bốn mảng như các đối số:

```
Sub WriteData (CNames() As String, CBalc() As Currency,
CDate() As Variant, CRegion() As Integer)
```

Hàm UBound() xây dựng sẵn sẽ trả lại chỉ số cao nhất đã được định nghĩa cho một mảng bất kỳ. Câu lệnh sau, có lẽ xuất hiện trong thủ tục con WriteData(), sẽ lưu chỉ số có thể có cao nhất cho mảng Cnames(), cho nên thủ tục con sẽ không cố truy xuất một chỉ số mảng ở ngoài giới hạn được định nghĩa:

```
HighSub = UBound(CNames)
```

Cảnh báo

Đừng quên rằng câu lệnh Call thật là lạ về các ngoặc đơn của đối số. Nếu sử dụng Call, bạn cũng phải bao quanh các đối số trong cặp dấu ngoặc đơn. Bạn có thể bỏ qua từ khóa Call, nhưng nếu thực hiện điều đó, nên bỏ qua cả cặp dấu ngoặc đơn. Sau đây một lệnh Call tương đương như vậy được minh họa trong Hình 16.2:

```
RecProc I, J, K ' No Call, no parens!
```

Tóm tắt

Giả sử bạn đang viết bộ chương trình nhằm mục đích kiểm kê và tìm khách hàng của một hiệu sách. Các chủ hiệu sách đòi hỏi người dùng nhập vào mã hạng mục mô tả loại đề mục sách cần tìm hoặc mua, và chương trình sẽ in mô tả của hạng mục đó. Việc mua sách được nhập vào hoặc khi một mục sách được tìm thấy qua việc kiểm kê. Nói cách khác, bạn sẽ hỏi người dùng về một mã hạng mục 1, 2, 3, 4, hay 5, và chương trình sẽ trả lại một mô tả trong biến thông báo chuỗi mô tả hạng mục đó. Lệnh Select Case sau sẽ làm việc:

```
Select Case CatCode
 Case 1:
 CatMessage = "Book"
 Case 2:
 CatMessage = "Magazine"
 Case 3:
 CatMessage = "Newspaper"
 Case 4:
 CatMessage = "Writing Supplies"
 Case 5:
 CatMessage = "Software"
 Case Else:
 CatMessage = "The category code is in error"
End Select
```

Vấn đề ở chỗ chiều dài của lệnh Select Case cần trong chương trình và cần bởi nhiều chương trình Visual Basic khác mà bạn đang viết. Bạn lưu trữ những mã lệnh này sẽ tốt hơn nhiều như một thủ tục Function đa dụng trong Ví dụ 16.4, và gọi hàm từ bất kỳ nơi nào khi cần đến mã lệnh, giống như sau:

```
CatDesc = DispCatCode$(CatCode) ' CatDesc is a string
```

Thủ tục cho phép bạn chuyển vào đối số mã hạng mục và trả về mô tả chuỗi phù hợp đối số đó. Ngay khi bạn lưu thủ tục trong một tập tin module, ứng dụng bất kỳ bạn thêm tập tin module đó vào sẽ có thể gọi hàm và nhận thông báo.

Ôn lại

Bằng cách chuyển đối số giữa thủ tục Function và thủ tục Subroutine, các thủ tục của bạn có thể chia sẻ dữ liệu cục bộ. Nếu một thủ tục không cần truy xuất tới một biến cục bộ khác, bạn sẽ không bao giờ chuyển biến cho thủ tục đó. Thông qua việc chỉ chuyển dữ liệu khi cần, bạn có thể truy xuất dữ liệu dựa trên nền tảng, chỉ các thủ tục cần truy xuất mới có thể lấy dữ liệu truy xuất. Nếu hết thảy biến của bạn là toàn cục, một thủ tục bất kỳ có thể thay đổi tình cờ một biến khác, và những lỗi logic như thế thường khó tìm.

Ví dụ 16.4. *Một thủ tục Function đa dụng mà bạn có thể gọi từ bất kỳ một thủ tục khác.*

```
1: Function DispCatCode$(CatCode As Integer)
2: Dim CatDesc As String
3: Select Case CatCode
4: Case 1:
5: CatMessage = "Book"
6: Case 2:
7: CatMessage = "Magazine"
8: Case 3:
9: CatMessage = "Newspaper"
10: Case 4:
11: CatMessage = "Writing Supplies"
12: Case 5:
13: CatMessage = "Software"
14: Case Else:
15: CatMessage = "The category code is in error"
16: End Select
17: DispCatCode = CatMessage ' Returns a description
18: End Function
```

Phân tích

Hàm DispCatCode$() có thể lưu trong một tập tin module ngoài cùng với nhiều thủ tục Function và Subroutine khác mà các ứng dụng của bạn luôn cần đến. Tập tin module hoạt động giống hộp công cụ với các công cụ (thủ tục) bạn có thể sử dụng (gọi) ở thời điểm bất kỳ.

Dòng 1 sẽ định nghĩa thủ tục là một hàm nhận đối số nguyên từ lệnh gọi. Dòng 2 định nghĩa một biến chuỗi cục bộ mà những lệnh Case sẽ sử dụng như một vùng lưu trữ tạm cho các mô tả thích hợp. Từ dòng 4 đến dòng 15 sau đó sẽ kiểm tra số nguyên được chuyển tới để xem liệu mã hạng mục là 1, 2, 3, 4, 5, hay là một giá trị khác sẽ đưa ra lỗi. Dòng 17 đảm bảo rằng thông báo thích hợp được trả lại cho thủ tục gọi.

THAM BIẾN VÀ THAM TRỊ

Khái niệm

Có hai cách nhận đối số chuyển bằng tham biến và tham trị. Phương thức bạn sử dụng xác định liệu thủ tục nhận có thể thay đổi các đối số để thay đổi đó vẫn còn hiệu quả sau khi thủ tục gọi chiếm lại điều khiển. Nếu bạn chuyển và nhận theo tham biến (phương thức mặc định), các biến cục bộ được chuyển của thủ tục gọi có thể thay đổi trong thủ tục nhận. Nếu bạn chuyển và nhận bằng tham trị, thủ tục gọi có thể truy xuất và thay đổi các đối số đã nhận của nó, nhưng những thay đổi này không để lại ảnh hưởng trong thủ tục gọi.

Subroutine và Function luôn luôn sử dụng các giá trị đã nhận của chúng và cũng thay đổi những đối số này. Nếu thủ tục nhận thay đổi một trong các đối số của nó, biến tương ứng trong thủ tục gọi cũng bị thay đổi. Vì vậy, khi thủ tục gọi chiếm lại điều khiển, (các) giá trị mà thủ tục gọi đã gửi như một đối số tới Subroutine được gọi có thể khác so với trước khi gọi.

Đối số chuyển bằng tham biến (theo địa chỉ – by address), nghĩa là đối số được chuyển có thể thay đổi bằng thủ tục nhận. Nếu bạn muốn thủ tục nhận không thể thay đổi các đối số của thủ tục gọi, bạn phải chuyển các đối số bằng tham trị (theo giá trị – by value). Để chuyển bằng tham trị, hãy đặt trước bất kỳ hay tất cả danh sách đối số nhận từ khóa ByVal, hay bao các đối số được chuyển trong cặp dấu ngoặc đơn.

Lời nhắc

Nhìn chung nhận các đối số bằng tham trị thì an toàn hơn bởi vì thủ tục gọi có thể chắc chắn rằng các giá trị được chuyển của nó sẽ không thay đổi bằng thủ tục nhận. Tuy nhiên, đôi khi bạn muốn thủ tục nhận thay đổi thường xuyên các giá trị được chuyển tới nó, và bạn sẽ nhận những đối số này dưới dạng tham biến (theo địa chỉ).

Tóm tắt

Ví dụ 16.5 sẽ chỉ ra hai thủ tục Subroutine. Thủ tục thứ nhất Changes(), sẽ nhận các đối số của nó bằng tham biến. Thủ tục thứ hai, NoChanges() nhận các đối số của nó bằng tham trị. Mặc dù cả hai thủ tục đều nhân các đối số của chúng cho 10, song thay đổi này chỉ ảnh hưởng đến các biến của thủ tục gọi, khi Change() được gọi, chứ không ảnh hưởng khi NoChanges() được gọi.

Ôn lại

Phương thức bạn gửi và nhận đối số sẽ xác định những thay đổi của một thủ tục nhận sẽ tồn tại ảnh hưởng trong bao lâu. Nếu thủ tục nhận thay đổi một hay nhiều đối số được nhận theo tham biến của nó, các thay đổi này sẽ duy trì ảnh hưởng khi thủ tục gọi chiếm lại điều khiển. Vì vậy, khi một thủ tục gọi chấp nhận các đối số được chuyển và nhận theo tham biến, hãy nhận thức rằng các giá trị được chuyển có thể sẽ khác khi điều khiển trả lại thủ tục gọi.

Ví dụ 16.5. *Một thủ tục nhận đối số theo kiểu tham chiếu và một thủ tục khác nhận đối số theo kiểu tham trị.*

```
1: Sub Changes (N As Integer, S As Single)
2: ' Receives arguments by address
3: N = N * 2 ' Double both
4: S = S * 2 ' arguments
5: ' When the calling routine regains control,
6: ' its two local variables will now be twice
7: ' as much as they were before calling this.
8: End Sub
9:
10: Sub NoChanges (ByVal N As Integer, ByVal S As Single)
11: ' Receives arguments by value
12: N = N * 2 ' Double both
13: S = S * 2 ' arguments
14: ' When the calling routine regains control,
15: ' its two local variables will not be
16: ' changed from their original values.
17: End Sub
```

Phân tích

Dòng 1 định nghĩa thủ tục Changes() để nhận hai đối số theo kiểu tham chiếu. Vì vậy, khi các dòng 3 và 4 nhân đôi hai giá trị này, các biến của thủ tục đang gọi cũng sẽ được nhân đôi.

Dòng 10 định nghĩa thủ tục NoChanges() nhận hai đối số của nó theo kiểu tham trị. Từ khóa ByVal báo cho Visual Basic biết không có vấn đề gì xảy ra với các đối số trong thủ tục NoChanges(), việc gọi các giá trị của thủ tục sẽ không bị ảnh hưởng bởi những thay đổi.

Các câu lệnh sau sẽ gọi chính xác hai thủ tục này:

```
Call Changes(N, S)
Call NoChanges(X, S)
```

Một lần nữa, các tên biến trong khi gọi thủ tục không phù hợp với tên biến nhận được trong danh sách đối số. Vì vậy, thủ tục gọi gởi biến X cục bộ của nó tới NoChanges(), sẽ tham khảo giá trị đó như trong thủ tục nhận. Nếu bạn muốn gửi dữ liệu tới mỗi thủ tục và chắc chắn rằng các biến đã gởi không bị thay đổi trong thủ tục nhận bằng kiểu tham chiếu, hãy đóng từng đối số được gửi ở câu được gửi trong cặp dấu ngoặc đơn như sau:

```
Call Changes( (N), (S) )
Call NoChanges( (X), (S) )
```

Mã lệnh thành phần ngoặc đơn quan trọng hơn bất kỳ một tham chiếu nào được chuyển và bảo đảm rằng các giá trị được chuyển sẽ giữ lại những giá trị ban đầu của nó sau khi các thủ tục được gọi hoàn thành.

Bài tập

Kiến thức tổng quát

1. Tên phạm vi của một biến được định nghĩa bên trong một thủ tục là gì?
2. Tên phạm vi của một biến được định nghĩa bằng cách sử dụng từ khóa Dim bên trong một module là gì?
3. Tên phạm vi của một biến được định nghĩa với từ khóa Global là gì?
4. Đúng hay Sai: Bạn có thể định nghĩa biến toàn cục bên trong một module của mẫu biểu.
5. Đúng hay Sai: Bạn có thể định nghĩa biến module bên trong một module của mẫu biểu.

6. Đúng hay Sai: Bạn có thể định nghĩa biến module bên trong một module không có mẫu biểu.
7. Loại phạm vi nào ít an toàn nhất ?
8. Loại phạm vi nào ít được dùng nhất?
9. Các giá trị trong CONSTANT.BAS được ấn định phạm vi như thế nào?
10. Từ khóa Const làm gì?
11. Tại sao bạn phải gán các giá trị ban đầu cho tên hằng khi định nghĩa tên hằng?
12. Bạn phải định nghĩa tất cả tên hằng ở đâu?
13. Đúng hay Sai: Ứng dụng của bạn có thể có hai biến khác nhau có cùng tên.
14. Điều gì sẽ xảy ra với biến cục bộ khi các thủ tục của chúng kết thúc?
15. Tên thủ tục gửi các đối số thủ tục khác là gì?
16. Tên thủ tục nhận các đối số từ thủ tục khác là gì?
17. Đúng hay Sai: Các hàm xây dựng sẵn không cần đối số.
18. Tại sao bạn phải xác định kiểu dữ liệu cho từng đối số trong danh sách đối số đang nhận?
19. Đúng hay Sai: Đôi khi bạn sử dụng lệnh Call để gọi các thủ tục Function.
20. Đúng hay Sai: Bạn có thể chuyển và nhận nhiều nhất một giá trị từ các thủ tục Function.
21. Có bao nhiêu cách để một thủ tục có thể nhận các biến?
22. Cho biết ý nghĩa của thuật ngữ *by address* đối với chương trình.
23. Cho biết ý nghĩa của thuật ngữ *by value* đối với chương trình.
24. Trình bày hai cách đảm bảo các đối số được chuyển bằng giá trị.
25. Khi nhận các mảng, hàm nào sẽ báo cho thủ tục nhận giá trị chỉ số cao nhất có thể có của mảng?

Viết mã lệnh...

26. Hãy định nghĩa hai biến toàn cục mà bạn có thể sử dụng để lưu giữ tên và tuổi một người.
27. Hãy định nghĩa hai hằng cục bộ lưu giữ số ngày trong tuần và số tháng trong năm.

Tìm lỗi kỹ thuật

28. Có gì không ổn với các định nghĩa biến toàn cục sau đây?

    A. Global Const X1 As Integer

    B. Global Const X2 As Integer = 19

    C. Global X3 = 19

    D. Global X4 As Integer = 19

29. An Huy cần viết một thủ tục Function chấp nhận một biến nguyên và một mảng chuỗi chứa 45 phần tử. An Huy nhận được lỗi với câu lệnh định nghĩa hàm sau đây. Bạn có thể khuyên anh ấy nên thực hiện những gì không ?

```
Function Report(Age As Integer, CoNames(45) As String)
```

Phần nâng cao

Hãy viết một thủ tục Function đa dụng chấp nhận một đối số chuỗi và trả lại một chuỗi chỉ chứa những ký tự khác nhau trong chuỗi đã nhận. Trả lại một chuỗi rỗng, "", nếu chuỗi nhận được chứa ít hơn hai ký tự.

Bài thực hành 8

Lập trình theo module

Tóm tắt

Chương này chỉ bạn cách chia chương trình thành từng phần mã lệnh riêng biệt trong mẫu biểu, thủ tục Subroutine, thủ tục Function và các tập tin module bên ngoài. Các thành phần khác nhau trong trình ứng dụng làm việc với nhau tạo nên chương trình mà người dùng sẽ thấy khi vận hành.

Bằng cách chia chương trình thành từng phần riêng biệt, bạn có thể tận dụng những công việc đã xây dựng, chẳng hạn sử dụng lại thủ tục Subroutine và thủ tục Function được viết trước đó. Bạn có thể định nghĩa tập tin cho các hằng toàn cục như CONSTANT.BAS. Khi dùng lại mã lệnh đã viết trước đó, bạn sẽ tiết kiệm được thời gian thiết kế chương trình làm cho quá trình dò tìm lỗi đơn giản hơn nhiều.

Trong chương này, bạn đã nắm bắt:

- Chương trình con là gì
- Thủ tục Subroutine khác thủ tục Function như thế nào
- Khi nào thì dùng tập tin module bên ngoài
- Tại sao Visual Basic hỗ trợ ba loại phạm vi biến
- Bạn chuyển các giá trị cục bộ từ thủ tục này sang thủ tục khác như thế nào

Mô tả chương trình



Hình P8.1. *Cửa sổ Project của Project8.*

Hình P8.1 mô tả cửa sổ Project PROJECT8.VBP. Như bạn sẽ thấy, project gồm một tập tin mẫu biểu chứa mẫu biểu, điều khiển và mã lệnh gắn liền với mẫu biểu cũng như tất cả thủ tục biến cố của điều khiển. Project cũng chứa tập tin lưu tên hằng PROJECT8.BAS, tập tin module bên ngoài được thiết kế cho trình ứng dụng này.

Mục đích của trình ứng dụng trong bài thực hành này là sử dụng tên hằng, thủ tục dùng chung, module bên ngoài và các điều khiển bạn đã làm việc trước khi gắn chúng với nhau trong chương trình.

Hoạt động chương trình

Khi bạn nạp và chạy PROJECT8.VBP, đầu tiên bạn sẽ thấy một hộp nhận dữ liệu yêu cầu giá trị từ 1 đến 100. Giá trị mặc định là 50 được sử dụng trong trường hợp người dùng muốn nhấn phím Enter để chấp nhận giá trị mặc định. Hộp nhận dữ liệu duy trì quá trình hiển thị cho đến khi người dùng nhập vào một số biến thiên trong khoảng từ 1 đến 100, hay nhấp Cancel hoặc OK để bỏ qua hộp nhận dữ liệu và chấp nhận giá trị mặc định bằng 50.

Hình P8.2. *Cửa sổ Form của Project8.*

Sau khi người dùng nhập vào một số hợp lệ, Hình P8.2 mô tả mẫu biểu của trình ứng dụng PROJECT8.VBP sẽ xuất hiện sau khi người dùng nhập vào một số hợp lệ. (Số được người dùng nhập trước khi mẫu biểu ở Hình P8.2 xuất hiện giá trị 47.)

Chương trình giả sử rằng số nhập vào của người dùng là một số thuộc cơ số 10 và cho biết nhiệt độ tính theo độ F. Nếu nhấp nút tùy chọn Base 16, bạn sẽ thấy số tương ứng với số người dùng nhập trong hệ thập lục phân ở mục Converted number:. Nhấp nút tùy chọn Base 8 để số người dùng nhập hiển thị trong khung Bases theo hệ bát phân (cơ số 8).

Khung Temperatures sẽ thay đổi nhiệt độ theo độ F của người dùng thành độ C và ngược lại phụ thuộc vào nút tùy chọn được chọn lựa.

Nếu người dùng nhấp nút lệnh Change Number, người dùng sẽ được yêu cầu nhập vào giá trị số mới.

Mã lệnh module bên ngoài

Ví dụ P8.1 trình bày danh sách mã lệnh đầy đủ cho module bên ngoài PROJECT8.BAS. Module bao gồm các tên hằng toàn cục cũng như hàm dùng chung nhận giá trị độ F và trả lại giá trị theo độ C tương ứng. Bạn có thể thêm tập tin module bên ngoài này vào trình ứng dụng bất kỳ bạn viết cần chuyển giá trị độ F thành độ C.

Ví dụ P8.1. *Tập tin module bên ngoài gồm các tên hằng của trình ứng dụng.*

```
Option Explicit

' Define global constants for the application

' Number base constants that match Index values
Global Const BASE10 = 10
Global Const BASE8 = 8
Global Const BASE16 = 16

' Temperature constants that match Index values
Global Const CELSIUS = 0
Global Const FAHRENHEIT = 1

```

```
14: Function GetCel (ByVal Faren As Integer)
15: ' Assumes that a Fahrenheit temperature
16: ' is passed. Returns that temp as Celsius
17: GetCel = (Faren + 40) * (5 / 9) – 40
18: End Function
```

Mô tả

1: Yêu cầu tất cả biến trong module phải được định nghĩa (hoặc các đối số được chuyển đến từ nơi khác).

2: Dòng trống giúp phân tách các phần trong mã lệnh.

3: Chú thích giúp giải thích mục đích thủ tục (General).

4: Dòng trống giúp phân tách các phần trong mã lệnh.

5: Chú thích giải thích các tên hằng tương ứng với thuộc tính chỉ số Index trong hai mảng điều khiển nút tùy chọn.

6: Định nghĩa tên hằng thay cho giá trị thuộc tính Index ứng với mục Base 10 trong mảng điều khiển nút tùy chọn. Thông qua cửa sổ Property, ba nút tùy chọn này có giá trị thuộc tính Index bằng 10, 8, và 16.

(6: Thay vì ghi giá trị Index, phần còn lại của chương trình sẽ sử dụng các tên hằng.)

7: Định nghĩa tên hằng thay cho giá trị thuộc tính Index ứng với mục Base 8 trong mảng điều khiển nút tùy chọn.

8: Định nghĩa tên hằng thay cho giá trị thuộc tính Index ứng với mục Base 16 trong mảng điều khiển nút tùy chọn.

9: Dòng trống giúp phân tách các phần mã lệnh.

10: Chú thích giúp giải thích mục đích mã lệnh kế tiếp.

11: Định nghĩa tên hằng thay cho giá trị 0 trong thuộc tính Index ứng với mục Celcius trong mảng điều khiển nút tùy chọn nhiệt độ. Từ cửa sổ Property, có thể thấy hai nút tùy chọn này có giá trị thuộc tính Index bằng 0 và 1.

12: Định nghĩa tên hằng thay cho giá trị 1 trong thuộc tính Index ứng với mục Fahrenheit trong mảng điều khiển nút tùy chọn.

13: Dòng trống phân tách thủ tục (General) với thủ tục Function.

14: Bắt đầu thủ tục Function dùng chung, nhận một đối số theo tham trị.

15: Chú thích giúp giải thích mục đích của hàm.

16: Tiếp tục chú thích trên dòng tiếp theo.

17: Tính giá trị trả về bằng cách gán giá trị nhiệt độ theo độ F đã được đổi cho tên hàm.

(17: Luôn trả lại một giá trị khi viết thủ tục Function.)

18: Kết thúc hàm.

Mã lệnh trong module mẫu biểu

Ví dụ P8.2 trình bày danh sách mã lệnh đầy đủ trong tập tin PROJECT8.VBP sử dụng module bên ngoài PROJECT8.BAS.

Ví dụ P8.2. *Mã lệnh module mẫu biểu điều khiển quá trình vận hành chương trình.*

```
Sub Form_Load ()
Call GetUserNum ' Stored in module file
optbase(BASE10).Value = True
optTemp(FAHRENHEIT).Value = True
' Trigger the event procedures
' for option button frames
Call optBase_Click(BASE10)
Call optTemp_Click(FAHRENHEIT)
End Sub
Sub GetUserNum ()
' A subroutine procedure that gets a
' number from 1 to 100 from the user
' and displays the number on the form
Dim UserNum As Variant
Do
UserNum = InputBox("Enter a number from 1 to 100", "Ask", "50")
If (UserNum = "") Then ' Check for Cancel
' Restore previous value
UserNum = lblUserNum.Caption
Exit Sub
End If
Loop While (UserNum < 1) Or (UserNum > 100)
lblUserNum.Caption = UserNum
End Sub
Sub optBase_Click (Index As Integer)
' Determines what base the converted
' number displays in the base frame
```

```
Select Case Index
Case BASE10:
' No change
lblBaseOut.Caption = lblUserNum.Caption
Case BASE16:
lblBaseOut.Caption = Hex$(lblUserNum.Caption)
Case BASE8:
lblBaseOut.Caption = Oct$(lblUserNum.Caption)
End Select
End Sub

Sub optTemp_Click (Index As Integer)
' Determines what temperature appears
Select Case Index
Case CELSIUS:
lblTempOut.Caption = GetCel(Val(lblUserNum.Caption))
Case FAHRENHEIT:
lblTempOut.Caption = lblUserNum.Caption
End Select
End Sub

Sub cmdChange_Click ()
' Asks the user once again for the number
' and calls appropriate click event
' procedures to update two frames
Call GetUserNum
optbase(BASE10).Value = True
optTemp(FAHRENHEIT).Value = True
Call optBase_Click(BASE10)
Call optTemp_Click(FAHRENHEIT)
End Sub

Sub cmdExit_Click ()
End
End Sub
```

Mô tả

1: Định nghĩa thủ tục sẽ thi hành ngay trước khi người dùng thấy mẫu biểu.

2: Gọi thủ tục Subroutine yêu cầu người dùng một số từ 1 đến 100.

3: Kích hoạt nút tùy chọn Base 10. Khi hiển thị mẫu biểu, nút tùy chọn Base 10 sẽ được chọn.

4: Kích hoạt nút tùy chọn Fahrenheit. Khi hiển thị mẫu biểu, nút tùy chọn Fahrenheit sẽ được chọn.

5: Dòng trống giúp phân tách các phần của chương trình.

6: Chú thích giải thích mã lệnh kế tiếp.

7: Tiếp tục chú thích.

8: Kích hoạt thủ tục biến cố Click ứng với các nút tùy chọn trong khung Bases. Biến cố này sẽ khởi tạo khung.

(8: Mã lệnh có thể kích hoạt thủ tục biến cố.)

9: Kích hoạt thủ tục biến cố Click ứng với các nút tùy chọn trong khung Fahrenheit. Biến cố này sẽ khởi tạo khung.

10: Kết thúc thủ tục biến cố chương trình con.

11: Dòng trống phân tách thủ tục Form_Load() với thủ tục Subroutine theo sau đó.

12: Định nghĩa thủ tục Subroutine yêu cầu người dùng nhập một số.

13: Chú thích giải thích mã lệnh kế tiếp.

14: Chú thích tiếp theo.

15: Chú thích tiếp theo.

16: Định nghĩa một biến Variant sẽ chặn kết quả của hộp nhập liệu.

17: Bắt đầu vòng lặp yêu cầu người dùng một số.

18: Hiển thị hộp nhập và đợi người dùng trả lời.

19: Không thay đổi số đã chọn trước của người dùng (hay thuộc tính mặc định nếu đây là lần đầu tiên người dùng được yêu cầu nhập một số) nếu người dùng nhấp nút lệnh Cancel.

(19: Luôn kiểm tra quá trình chọn nút lệnh Cancel của người dùng.)

20: Chú thích giải thích mã lệnh kế tiếp.

21: Đặt số trở lại giá trị hiện hành của nó.

22: Kết thúc sớm thủ tục Subroutine.

23: Kết thúc lệnh If kiểm tra quá trình nhấn nút lệnh Cancel.

24: Duy trì quá trình yêu cầu cho đến khi người dùng nhập vào giá trị số hợp lệ trong vùng được yêu cầu.

25: Người dùng đã nhập vào giá trị số hợp lệ trong hộp nhập, cho nên sẽ hiển thị kết quả.

26: Kết thúc thủ tục Subroutine.

27: Dòng trống phân tách thủ tục GetUserNum() với thủ tục biến cố sau đó.

28: Định nghĩa thủ tục biến cố cho mảng điều khiển nút tùy chọn Bases.

29: Chú thích giải thích mục đích thủ tục biến cố.

30: Chú thích tiếp tục trên dòng tiếp theo.

31: Đối số Index có thể là một trong ba giá trị sau: 10, 16, hay 8, như khi đặt ba giá trị Index trong cửa sổ Properties. Kiểm tra giá trị trong thuộc tính Index để xác định nút tùy chọn nào để đáp ứng.

32: Nếu người dùng đã nhấp nút tùy chọn Base 10...

33: Chú thích mô tả mã lệnh kế tiếp.

34: Nhãn trong khung Bases phù hợp với số người dùng đã nhập bởi vì cả hai đều tính theo hệ thập phân.

35: Nếu người dùng nhấp nút tùy chọn Base 16...

36: Hiển thị số người dùng đã nhập như một chuỗi số trong hệ thập lục phân.

37: Nếu người dùng nhấp nút tùy chọn Base 8...

38: Hiển thị số người dùng đã nhập như một chuỗi số trong hệ bát phân.

39: Kết thúc lệnh Select Case.

40: Kết thúc thủ tục biến cố.

41: Dòng trống phân tách hai thủ tục biến cố.

42: Định nghĩa thủ tục biến cố cho mảng điều khiển nút tùy chọn Temperatures.

43: Chú thích giải thích mục đích thủ tục biến cố.

44: Đối số Index có thể là một hay hai giá trị : 0 hoặc 1, khi đặt hai giá trị Index trong cửa sổ Properties. Kiểm tra giá trị trong thuộc tính Index để xác định nút tùy chọn nào để đáp ứng.

45: Nếu người dùng nhấp nút tùy chọn Celsius...

46: Hiển thị số người dùng đã nhập như nhiệt độ tính theo độ C.

47: Nếu người dùng nhấp nút tùy chọn Fahrenheit...

48: Nhãn trong khung Temperature phù hợp với số người dùng đã nhập bởi vì cả hai cùng được biểu thị theo độ F.

49: Kết thúc lệnh Select Case.

50: Kết thúc thủ tục biến cố.

51: Dòng trống phân tách các thủ tục biến cố.

52: Định nghĩa thủ tục biến cố sẽ thi hành khi người dùng nhấp Change Number.

53: Chú thích giải thích mục đích thủ tục biến cố.

54: Tiếp tục chú thích dòng trước.

55: Tiếp tục chú thích dòng trước.

56: Gọi thủ tục Subroutine để lấy một số từ người dùng. Không có đối số nào được yêu cầu.

(56: Nếu thủ tục Subroutine không đòi hỏi đối số nào, đừng sử dụng cặp dấu ngoặc đơn.)

57: Đặt nút tùy chọn Base 10 làm nút mặc định trong khung Bases.

58: Đặt nút tùy chọn Fahrenheit làm nút mặc định trong khung Temperatures.

59: Kích hoạt thủ tục biến cố nhấp nút tùy chọn Base 10 mặc định để khung Bases hiển thị số vừa được người dùng nhập.

60: Kích hoạt thủ tục biến cố nhấp nút tùy chọn Fahrenheit để khung Temperatures hiển thị số vừa được người dùng nhập.

61: Kết thúc thủ tục biến cố.

62: Dòng trống phân cách thủ tục biến cố.

63: Định nghĩa thủ tục biến cố với nút lệnh Exit.

64: Kết thúc chương trình khi biến cố này kích hoạt.

65: Kết thúc thủ tục biến cố.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Chương kế tiếp sẽ giải thích cách bổ sung truy xuất tập tin vào trình ứng dụng để bạn có thể lưu và nạp dữ liệu lâu dài trong các tập tin đĩa.

Chương IX

Bài 17

Các điều khiển tập tin

- **Hộp thoại File**
- **Điều khiển tập tin**
- **FILESEL.VBP quản lý quá trình chọn tập tin**

Bài này mô tả cách truy xuất tập tin trong các trình ứng dụng Visual Basic. Dữ liệu lưu trong tập tin bao giờ vẫn cố định hơn dữ liệu lưu trong biến. Giá trị biến dễ thay đổi. Ví dụ, nếu muốn biết thông tin về khách hàng, bạn phải lưu thông tin đó trên đĩa. Còn như đã giữ thông tin như thế trong các biến mảng, bạn sẽ không bao giờ thoát khỏi Visual Basic hay tắt máy tính bởi vì lúc đó bạn sẽ mất tất cả thông tin!

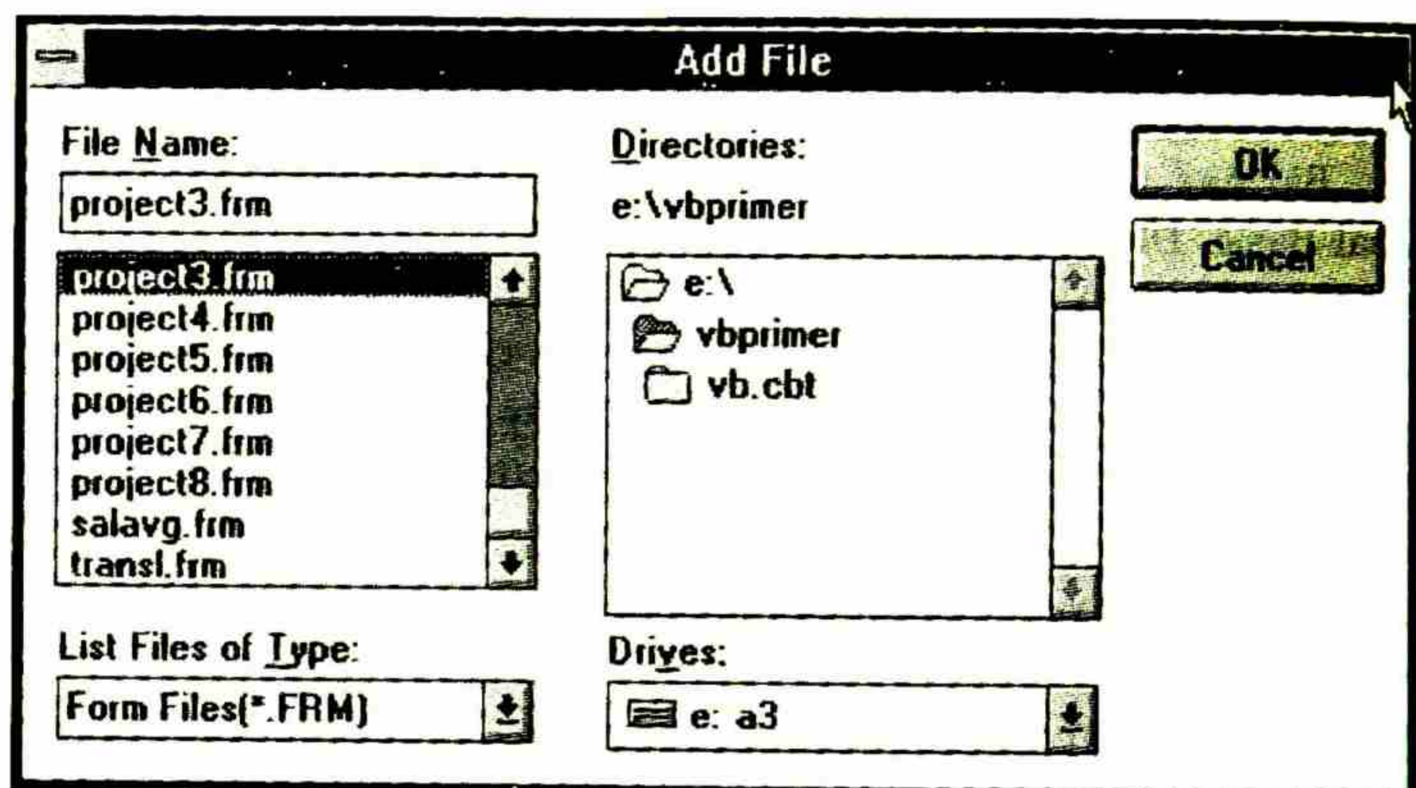
Dữ liệu bên trong các biến cần cho việc xử lý, tính toán, in ấn và lưu trữ, nhưng không lưu trữ lâu dài được. Trước khi truy xuất dữ liệu lưu trong các tập tin, bạn phải có khả năng tìm dữ liệu lưu trên đĩa. Bài này sẽ chỉ cho bạn cách xây dựng các hộp thoại quản lý tập tin thông thường cho phép người dùng chọn những tập tin được lưu trên đĩa. Phụ thuộc vào trình ứng dụng, có lẽ bạn cần người dùng chọn một tên tập tin, và việc trình bày hộp thoại File trong bài sẽ cung cấp những công cụ cần để hiển thị danh sách tập tin ở nơi người dùng có thể chọn. Sau khi người dùng chọn hay nhập vào một tên tập tin, bạn cần sử dụng lệnh và hàm nhập xuất (I/O) tập tin được đề cập trong bài tiếp theo để truy xuất dữ liệu trong tập tin.

HỘP THOẠI File

Khái niệm

Có nhiều cách lấy thông tin tên tập tin từ người dùng. Cách phổ biến nhất là xây dựng một hộp thoại cho phép người dùng chọn một tập tin, thư mục và ổ đĩa từ danh sách bạn cung cấp.

Hình 17.1. mô tả hộp thoại Add File thông thường nếu chọn lệnh Add File từ Visual Basic. Bạn đã thấy các hộp thoại Open File tương tự trong nhiều ứng dụng Windows. Điều quan tâm về các hộp thoại File là người dùng có thể chọn tập tin từ danh sách tên tập tin, thư mục, và ổ đĩa. Mặc dù bạn có thể yêu cầu người dùng tên tập tin bằng cách dùng hộp nhận dữ liệu, nhưng việc cho phép người dùng chọn từ danh sách nghĩa là người dùng có thể thấy các tập tin nhanh và chính xác hơn.



Hình 17.1. *Hộp thoại cho phép người dùng chọn tên tập tin, thư mục và ổ đĩa.*

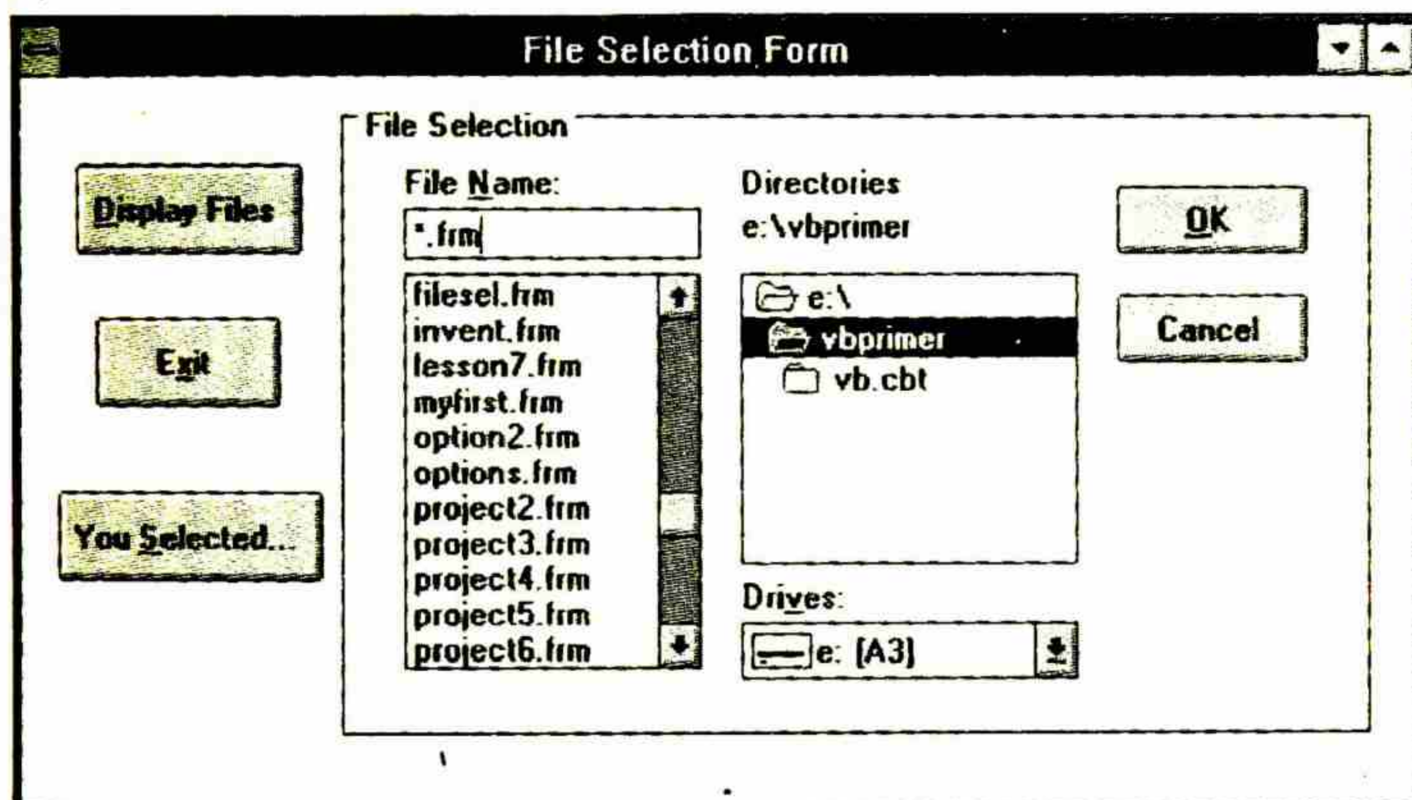
Các hộp thoại không hơn gì những mẫu biểu thứ hai thêm vào cửa sổ chính của trình ứng dụng. Hộp danh sách, nút lệnh, và hộp nhập đặt trên cửa sổ Form chính cũng không hơn gì những điều khiển đặt trên mẫu biểu thứ hai.

Điều khiển khung (Frame) đưa ra các khung có thể thiết đặt những điều khiển chọn tập tin trên đó chẳng hạn như hộp danh sách, nút lệnh và hộp nhập. Điều khiển khung không hoàn toàn giống hộp thoại thật bởi vì người dùng khó lòng định lại kích cỡ hay di chuyển. Tuy nhiên, điều cơ bản của khung là cho phép hiển thị những điều khiển chọn tập tin khi cần.

Ghi chú

Về mặt kỹ thuật, bạn có thể viết một điều khiển khung với các thuộc tính cho phép người dùng di chuyển khung, nhưng phần lớn, người dùng không nên thực hiện những khả năng đó trên các điều khiển. Người dùng dễ dàng che khuất các điều khiển khác và làm hỏng thiết kế của trình ứng dụng.

Hình 17.2 minh họa khung chọn tập tin mà bạn sẽ học cách tạo và sử dụng trong bài này. Khung có cùng chức năng như hộp thoại được minh họa trong Hình 17.1. Sau khi người dùng chọn một tập tin, bạn có thể lập trình khung để che khuất màn hình như khi che khuất các hộp thoại chọn tập tin.



Hình 17.2. *Một khung chọn tập tin bắt chước hộp thoại chọn tập tin.*

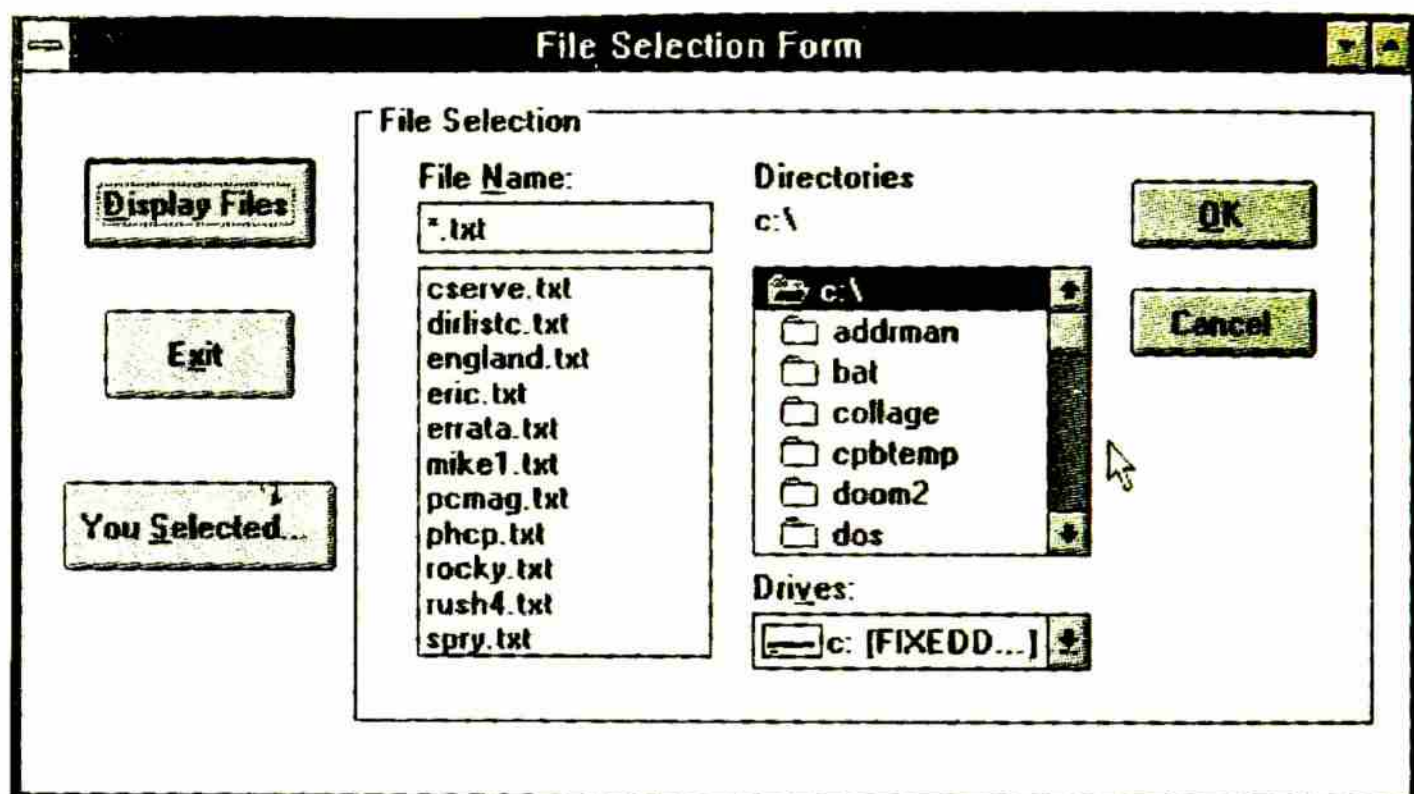
Thực tế, khung chọn tập tin dễ sử dụng hơn hộp thoại chọn tập tin bởi vì khung kết hợp hộp File Type với hộp nhập File Name. Khi người dùng nhập vào loại tập tin, chẳng hạn như *.frm trong hộp nhập File Name, khung sẽ cập nhật ngay lập tức hộp danh sách tên tập tin để chỉ hiện lại những tập tin phù hợp với loại đó.

Tóm tắt

Với trình ứng dụng được minh họa trong Hình 17.3, hãy chọn các ổ đĩa và thư mục khác nhau đồng thời chú ý rằng cách danh sách tập tin sẽ cập nhật để làm mới việc thay đổi. Cuối bài sẽ mô tả các thành phần quan trọng của trình ứng dụng này.

Ôn lại

Bạn có thể mô phỏng các mẫu biểu phụ bằng cách sử dụng điều khiển khung (Frame) để giữ các điều khiển bắt chước hộp thoại. Bài này sẽ tập trung vào việc giải thích project FILESEL.VBP để bạn hiểu cách xây dựng các khung chọn tập tin cho người dùng, và bạn sẽ biết cách truy xuất thông tin được chọn sau khi người dùng hoàn thành việc chọn một tập tin.



Hình 17.3. *Project FILESEL.VBP với khung chọn tập tin trên màn hình.*

Mục "Điều khiển tập tin" sẽ giải thích cách dùng các điều khiển tập tin. Mỗi điều khiển tập tin sẽ đưa ra một loại hộp danh sách truy xuất tập tin khác nhau. Hình 17.3 bao gồm các ô gọi đến ba hộp danh sách được đưa ra bởi ba điều khiển tập tin.

Phân tích

Ứng dụng FILESEL.VBP minh họa cách xây dựng khung chọn tập tin. Trình ứng dụng không làm gì nhiều hơn việc hiển thị khung chọn tập tin khi người dùng nhấp nút lệnh Display Files, và khi người dùng chọn một tập tin, trình ứng dụng sẽ xóa khung và hiển thị nút lệnh mới có nhãn "You Selected... " để người dùng có thể thấy tập tin, đường dẫn và ổ đĩa của tập tin được chọn từ khung trước đó.

Trình ứng dụng đặc biệt không những cho phép người dùng chọn các tập tin mà còn nhập vào những tên tập tin mới. Nói cách khác, trình ứng dụng này tạo nền tảng cho khung cho phép người dùng mở các tập tin đã tồn tại. Nếu bạn muốn cho người dùng khả năng nhập một tên không có trong danh sách, bạn sẽ phải sửa đổi trình ứng dụng để tìm trong hộp nhập File Name tên tập tin mới và hợp lệ.

Chương trình FILESEL.VBP sẽ sử dụng thuộc tính Visible của khung, thiết đặt thuộc tính thành True cho việc hiển thị khung và thành False khi khung không phải xuất hiện trên màn hình.

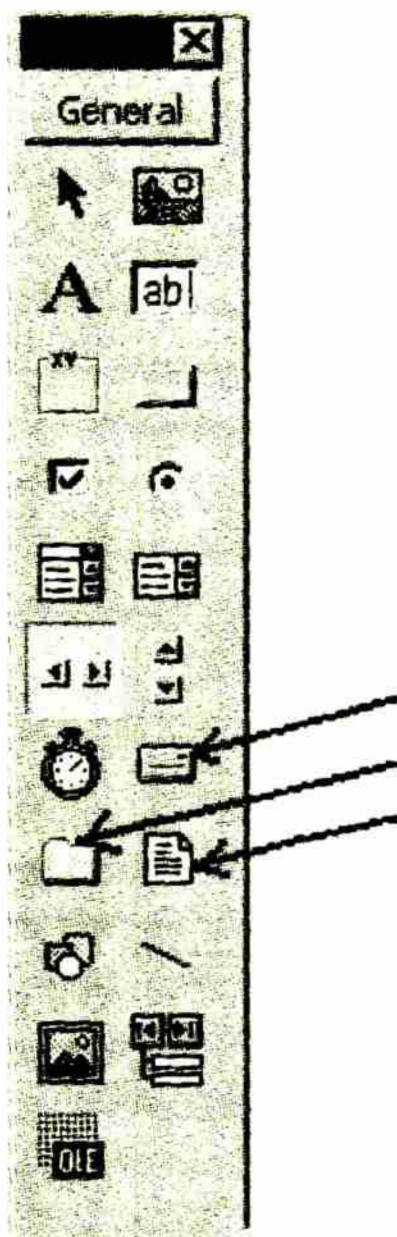
ĐIỀU KHIỂN TẬP TIN

Khái niệm

Visual Basic có ba điều khiển tập tin (File) mà khung chọn tập tin sẽ quản lý.

Hình 17.4 mô tả cửa sổ Toolbox cho thấy vị trí của ba điều khiển tập tin. Như bạn đã biết, quá trình chọn một điều khiển tập tin không chỉ chọn tên tập tin mà còn chọn thư mục và ổ đĩa. Khi đặt các điều khiển tập tin này trên khung, hãy nhớ rằng bạn phải vẽ từng điều khiển bằng cách kéo mouse (được giải thích đầy đủ hơn trong Bài 12 Chương 6) để mỗi điều khiển luôn theo khung khi mã lệnh của bạn di chuyển, che giấu hay hiển thị khung.

Sử dụng các điều khiển liên quan đến tập tin đòi hỏi bạn gắn chúng với nhau qua mã lệnh. Nói cách khác, khi người dùng thay đổi tên ổ đĩa và tên đường dẫn, danh sách tập tin phải thay đổi cùng với quá trình hiển thị đường dẫn các tập tin trên ổ đĩa được chọn gần nhất.



Hình 17.4. *Các điều khiển tập tin trên cửa sổ Toolbox.*

Bảng 17.1 liệt kê các thuộc tính sẽ xuất hiện trong cửa sổ Properties của điều khiển hộp danh sách ổ đĩa (Drive List Box). Các thuộc tính tương tự như thuộc tính của những điều khiển khác mà bạn đã biết. Điều khiển hộp danh sách ổ đĩa có chứa danh sách các ổ đĩa có hiệu lực trên hệ thống đang chạy chương trình. Tương tự điều khiển hộp danh sách bất kỳ, người dùng sẽ chọn từ hộp danh sách ổ đĩa lúc chạy, cho nên bạn không thể đặt một ổ đĩa mặc định trong danh sách cho đến khi chạy chương trình.

Bảng 17.1. *Các thuộc tính hộp danh sách ổ đĩa*

Thuộc tính	Diễn giải
BackColor	Xác định màu nền của hộp danh sách ổ đĩa, được chọn như một mã màu thập lục phân hay từ một bảng màu.
DragIcon	Chứa biểu tượng sẽ xuất hiện khi người dùng kéo hộp danh sách ổ đĩa trên mẫu biểu. (Bạn ít khi cho phép người dùng di chuyển hộp danh sách ổ đĩa, cho nên xác lập thuộc tính Drag... thường không thích hợp.)
DragMode	Có giá trị hoặc bằng 1 ứng với các yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển) hoặc giá trị 0 (mặc định) cho việc kéo mouse tự động, nghĩa là người dùng không thể kéo hộp danh sách ổ đĩa nhưng bạn, thông qua mã lệnh, có thể khởi tạo việc kéo nếu cần.
Enabled	Xác định hộp danh sách ổ đĩa có thể đáp ứng các biến cố. Nếu thiết đặt là True (mặc định), hộp danh sách ổ đĩa có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ kết thúc việc xử lý biến cố của điều khiển riêng biệt đó.
FontBold	Giữ giá trị True (mặc định) nếu tên ổ đĩa hiển thị ở dạng chữ đậm; ngược lại là False.
FontItalic	Giữ giá trị True (mặc định) nếu tên ổ đĩa hiển thị ở dạng chữ nghiêng; ngược lại là False.
FontName	Chứa các kiểu chữ của tên ổ đĩa trong hộp danh sách ổ đĩa. Thông thường, bạn sẽ sử dụng tên một phông chữ TrueType trong Windows.
FontSize	Kích cỡ tính theo point, của phông chữ dùng cho tên ổ đĩa.

FontStrikethru	Giá trị True (mặc định) nếu các tên hiển thị với những ký tự bị gạch ngang (ký tự có đường gạch ngang qua chúng); ngược lại là False.
FontUnderline	Giá trị True (mặc định) nếu các tên ổ đĩa hiển thị có gạch dưới; ngược lại là False.
ForeColor	Xác định màu ký tự trong tên ổ đĩa.
Height	Chứa chiều cao tính theo twip của hộp danh sách ổ đĩa.
HelpContextID	Nếu bạn thêm trợ giúp cảm ngữ cảnh cao cấp vào ứng dụng của bạn, HelpContextID sẽ cung cấp một số nhận biết văn bản trợ giúp.
Index	Nếu hộp danh sách ổ đĩa là thành phần của một mảng điều khiển, thuộc tính Index cung cấp số chỉ số cho từng hộp danh sách ổ đĩa riêng biệt. (Xem Chương 6.)
Left	Khoảng cách twip tính từ biên trái cửa sổ Form tới biên trái hộp danh sách ổ đĩa.
MousePointer	Cho biết hình dạng con trỏ mouse khi người dùng di chuyển con trỏ mouse trên hộp danh sách ổ đĩa. Các giá trị có thể từ 0 đến 12 và trình bày những hình dạng con trỏ mouse khác nhau. (Xem Chương 12.)
Name	Cho biết tên điều khiển. Mặc định, Visual Basic sẽ tự đặt các tên Drive1, Drive2, và ... khi thêm lần lượt các hộp danh sách ổ đĩa vào mẫu biểu.
TabIndex	Xác định thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần thêm điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm (focus) bằng cách thay đổi giá trị thuộc tính TabIndex của điều khiển. Không có hai điều khiển nào trên cùng mẫu biểu có cùng giá trị TabIndex.
TabStop	Nếu là True, xác định người dùng có thể nhấn phím Tab để di chuyển tiêu điểm tới hộp danh sách ổ đĩa. Nếu là False, hộp danh sách ổ đĩa không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Đây là thuộc tính dành cho lập trình viên xác định chú thích được áp dụng cho hộp danh sách ổ đĩa (Drive List Box).
Top	Khoảng cách tính bằng twip từ cạnh trên cùng hộp danh sách ổ đĩa tới cạnh trên cùng mẫu biểu.
Visible	Có giá trị True hoặc False, xác định người dùng có thể thấy (và sử dụng) hộp danh sách ổ đĩa.
Width	Độ rộng twip của hộp danh sách ổ đĩa.

Lúc chạy chương trình, bạn có thể thiết đặt thuộc tính Drive của hộp danh sách ổ đĩa. Thuộc tính Drive bao gồm tên ổ đĩa cũng như nhãn đĩa. Nếu chỉ muốn ký tự tên ổ đĩa, hãy sử dụng hàm Left$() để lấy ký tự ổ đĩa ở bên trái trong thuộc tính Drive của hộp danh sách ổ đĩa.

Bảng 17.2 liệt kê tất cả thuộc tính xuất hiện trong cửa sổ Properties của điều khiển hộp danh sách tập tin (File List Box). Nhiều thuộc tính giống như các điều khiển khác, nhưng cũng có một vài thuộc tính riêng quan trọng mà bạn sẽ điều khiển bằng cách dùng mã lệnh Visual Basic.

Bảng 17.2. *Các thuộc tính hộp danh sách tập tin*

Thuộc tính	Mô tả
Archive	Nếu là True (mặc định) xác định danh sách tập tin bao gồm cả tập tin lưu trữ (back up). Nếu là False, các tập tin mà bit lưu trữ (archive) của nó không được thiết đặt sẽ không hiển thị.
BackColor	Màu nền của hộp danh sách tập tin, được chọn như mã lệnh màu trong hệ thập lục phân hay từ bảng màu.
DragIcon	Giữ biểu tượng sẽ hiển thị khi người dùng kéo hộp danh sách tập tin trên mẫu biểu. (Hiếm khi bạn cho phép người dùng di chuyển hộp danh sách tập tin, cho nên các xác lập thuộc tính Drag... thường không thích hợp.)
DragMode	Hoặc bằng 1 cho biết yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển) hay bằng 0 (mặc định) để kéo mouse tự động, nghĩa là người dùng không thể kép hộp danh sách tập tin nhưng bạn có thể khởi đầu quá trình kéo thông qua mã lệnh nếu cần.
Enabled	Nếu định là True (mặc định), sẽ xác định hộp danh sách tập tin có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng quá trình xử lý biến cố của điều khiển đó.
FontBold	Giá trị True (mặc định) thì tên tập tin sẽ hiển thị ở dạng chữ đậm, ngược lại là False.
FontItalic	Giá trị True (mặc định) thì tên tập tin sẽ hiển thị ở dạng chữ nghiêng; ngược lại là False.

FontName	Tên của kiểu chữ cho tên tập tin trong hộp danh sách tập tin. Thông thường, bạn sẽ dùng tên phông chữ TrueType trong Windows.
FontSize	Kích cỡ phông chữ tính theo point dùng cho các tên tập tin.
FontStrikethru	Giá trị True (mặc định) nếu tên tập tin hiển thị với các ký tự gạch ngang (ký tự có đường gạch ngang qua); ngược lại là False.
FontUnderline	Giá trị True (mặc định) nếu tên tập tin hiển thị với các ký tự được gạch dưới; ngược lại là False.
ForeColor	Xác định màu các ký tự tên tập tin.
Height	Chiều cao tính theo twip của hộp danh sách tập tin.
HelpContextID	Nếu thêm trợ giúp cảm ngữ cảnh cao cấp vào trình ứng dụng, thuộc tính HelpContextID cung cấp một số xác định văn bản trợ giúp.
Hidden	Giá trị True, xác định danh sách tập tin bao gồm cả tập tin có thuộc tính ẩn. Nếu là False (mặc định), các tập tin có thuộc tính ẩn sẽ không hiển thị.
Index	Nếu hộp danh sách tập tin là thành phần của mảng điều khiển, thuộc tính Index sẽ cung cấp số chỉ số cho từng hộp danh sách tập tin riêng biệt (Xem Chương 6.)
Left	Khoảng cách twip từ biên trái cửa sổ Form tới biên trái hộp danh sách tập tin.
MousePointer	Cho biết hình dạng con trỏ mouse sẽ thay đổi khi người dùng di chuyển con trỏ mouse trên hộp danh sách tập tin. Các giá trị có thể có từ 0 đến 12 và đại diện cho những hình dạng con trỏ mouse khác nhau (xem Chương 12).
MultiSelect	Nếu thuộc tính MultiSelect bằng 0–None (mặc định), người dùng có thể chỉ chọn một tên tập tin. Nếu là 1–Simple, người dùng có thể chọn nhiều tên tập tin bằng cách nhấp mouse hay nhấn phím spacebar trên các tên tập tin trong danh sách. Nếu là 2–Extended, người dùng có thể chọn nhiều tên tập tin bằng cách dùng Shift+nhấp mouse và Shift+mũi tên để mở rộng quá trình chọn tên tập tin được chọn với tên tập tin hiện hành, dùng Ctrl+click để chọn hoặc bỏ chọn một tên tập tin khỏi danh sách.

Name	Tên điều khiển. Mặc định, Visual Basic sẽ tự phát sinh các tên File1, File2, và ... khi bạn thêm lần lượt các hộp danh sách tập tin vào mẫu biểu.
Normal	Nếu thuộc tính bằng True (mặc định), danh sách tập tin bao gồm các tập tin có thuộc tính Normal. Nếu là False, các tập tin có thuộc tính Archive sẽ không hiển thị.
Pattern	Cho biết các mẫu chọn tập tin. * cho biết chọn tất cả tập tin, ? nghĩa là mọi ký tự tại vị trí của nó. *.txt nghĩa là hết thảy tập tin có phần mở rộng là txt, và sales??.dat nghĩa là các tập tin có 5 ký tự đầu là sales, ký tự thứ 6 và 7 của tên tập tin là bất kỳ và phần mở rộng phải là dat.
ReadOnly	Giá trị True (mặc định) cho biết danh sách tập tin bao gồm cả tập tin có thuộc tính chỉ đọc. Vì vậy, danh sách tập tin sẽ hiển thị các tập tin này nhưng không thể thay đổi. Nếu là False, các tập tin có thuộc tính chỉ đọc sẽ không được hiển thị.
System	Giá trị True nếu danh sách tập tin bao gồm tập tin có thuộc tính hệ thống (System). Nếu là False (mặc định), các tập tin có thuộc tính hệ thống sẽ không hiển thị.
TabIndex	Xác định thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi giá trị TabIndex của điều khiển. Không có hai điều khiển nào trên cùng mẫu biểu có thể có cùng giá trị TabIndex.
TabStop	Giá trị True nếu người dùng có thể nhấn phím Tab để di chuyển tiêu điểm đến hộp danh sách tập tin. Nếu là False, hộp danh sách tập tin không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Thuộc tính này dành cho lập trình viên xác định chú thích áp dụng cho hộp danh sách tập tin.
Top	Khoảng cách twip từ cạnh trên cùng hộp danh sách tập tin tới cạnh trên cùng mẫu biểu.
Visible	Hoặc bằng True hoặc bằng False, cho biết người dùng có thể thấy và sử dụng hộp danh sách tập tin.
Width	Độ rộng tính theo twip của hộp danh sách tập tin.

Bảng 17.3 liệt kê các thuộc tính sẽ xuất hiện trong cửa sổ Properties của điều khiển hộp danh sách thư mục (Dir List Box). Nhiều thuộc tính giống thuộc tính của các điều khiển khác. Như điều khiển danh sách tập tin, bạn có thể đặt và chọn các giá trị từ hộp danh sách thư mục lúc chạy chương trình.

Bảng 17.3. *Các thuộc tính hộp danh sách thư mục*

Thuộc tính	Mô tả
BackColor	Xác định màu nền của hộp danh sách thư mục, được chọn như một mã màu trong hệ thập lục phân hay từ bảng màu.
DragIcon	Xác định biểu tượng sẽ xuất hiện khi người dùng kéo hộp danh sách thư mục trên mẫu biểu. (Hiếm khi bạn để cho người dùng di chuyển hộp danh sách thư mục, cho nên các xác lập thuộc tính Drag... thường không thích hợp.)
DragMode	Giá trị 1 ứng với yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong quá trình kéo điều khiển), giá trị 0 (mặc định) cho biết quá trình kéo mouse tự động, nghĩa là người dùng không thể kéo hộp danh sách thư mục, nhưng bạn thông qua mã lệnh có thể bắt đầu quá trình kéo nếu cần.
Enabled	Xác định hộp danh sách thư mục có thể đáp ứng các biến cố nếu thiết đặt là True (mặc định). Nếu định là False, Visual Basic sẽ dừng quá trình xử lý biến cố cho điều khiển riêng biệt đó.
FontBold	Giá trị True (mặc định) thì các tên thư mục sẽ hiển thị ở dạng chữ đậm, ngược lại là False.
FontItalic	Giá trị True (mặc định) thì tên thư mục sẽ hiển thị ở dạng chữ nghiêng; ngược lại là False.
FontName	Tên kiểu chữ của tên thư mục trong hộp danh sách thư mục. Thông thường, bạn sẽ dùng tên phông chữ TrueType trong Windows.
FontSize	Xác định kích cỡ theo point của phông chữ dùng cho tên thư mục.
FontStrikethru	Giá trị True (mặc định) sẽ hiển thị ký tự gạch ngang trên các tên thư mục (nghĩa là các ký tự có đường gạch ngang qua); ngược lại là False.

FontUnderline	Giá trị True (mặc định) sẽ hiển thị ký tự gạch dưới tên thư mục; ngược lại là False.
ForeColor	Xác định màu ký tự tên thư mục.
Height	Chiều cao tính theo twip của hộp danh sách thư mục.
HelpContextID	Nếu thêm trợ giúp cảm ngữ cảnh cao cấp vào trình ứng dụng, thuộc tính HelpContextID sẽ cung cấp một số nhận biết văn bản trợ giúp.
Index	Nếu hộp danh sách thư mục là thành phần của mảng điều khiển, thuộc tính Index sẽ cung cấp số chỉ số cho từng hộp danh sách thư mục riêng biệt. (Xem Chương 6.)
Left	Khoảng cách twip tính từ biên trái cửa sổ Form tới biên trái hộp danh sách thư mục.
MousePointer	Xác định hình dạng con trỏ mouse sẽ thay đổi nếu người dùng di chuyển con trỏ mouse trên hộp danh sách thư mục. Các giá trị có thể từ 0 đến 12 và đại diện cho những hình dạng con trỏ mouse khác nhau. (Xem Chương 12.)
Name	Chứa tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên Dir1, Dir2, và ... khi thêm lần lượt các hộp danh sách thư mục vào mẫu biểu.
TabIndex	Xác định thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần thêm một điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi giá trị thuộc tính TabIndex của điều khiển. Không có hai điều khiển trên cùng mẫu biểu có cùng giá trị TabIndex.
TabStop	Nếu là True sẽ xác định người dùng có thể nhấn Tab để di chuyển tiêu điểm tới hộp danh sách thư mục. Nếu là False, hộp danh sách thư mục không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Thuộc tính này dành cho lập trình viên xác định chú thích áp dụng cho hộp danh sách thư mục.
Top	Khoảng cách tính theo twip từ cạnh trên cùng hộp danh sách thư mục tới cạnh trên cùng mẫu biểu.
Visible	Có giá trị True hoặc False, cho biết người dùng có thể thấy và sử dụng hộp danh sách thư mục.
Width	Độ rộng tính bằng twip của hộp danh sách thư mục.

Lưu ý

Trong thời gian chạy chương trình, bạn có thể định thuộc tính Path của hộp danh sách thư mục. Thuộc tính Path bao gồm tên ổ đĩa cùng với đường dẫn đầy đủ.

Đa số các điều khiển liên quan đến tập tin đều chung. Tuy nhiên, bạn sẽ phải cập nhật không ngừng các giá trị thuộc tính khóa của những điều khiển liên quan đến tập tin này khi người dùng chọn các giá trị khác nhau từ điều khiển.

Ví dụ, khi người dùng chọn tên ổ đĩa khác, bạn phải cập nhật lại danh sách thư mục trong điều khiển danh sách thư mục cũng như danh sách tên tập tin trong điều khiển danh sách tập tin. Khi người dùng chọn thư mục khác, bạn phải cập nhật lại danh sách tên tập tin trong điều khiển danh sách tập tin nhưng không thay đổi điều khiển danh sách ổ đĩa. Việc duy trì tính đồng nhất của các điều khiển tập tin này là yêu cầu đáng chú ý duy nhất khi sử dụng chúng. Phần tiếp theo sẽ giải thích cách ứng dụng FILESEL.VBP sử dụng mã lệnh để gắn các điều khiển tập tin này với nhau.

Lưu ý

Biến cố có hiệu lực với điều khiển tập tin hoàn toàn giống với biến cố bạn đã gặp. Thủ tục biến cố lập trình phổ biến nhất là Click và DblClick. Thêm vào đó, hộp danh sách ổ đĩa và thư mục sẽ hỗ trợ thủ tục biến cố Change sẽ thi hành khi người dùng thay đổi ổ đĩa hoặc thư mục.

Củng cố

Mặc dù có nhiều thuộc tính dành cho các điều khiển liên quan đến tập tin hoàn toàn giống với những điều khiển khác, song vẫn có một vài thuộc tính khóa xác định cách chọn tên tập tin mà người dùng có thể thực hiện. Đặc biệt là các thuộc tính sau:

- Thuộc tính Drive của điều khiển hộp danh sách ổ đĩa (Drive List Box)
- Thuộc tính Path của điều khiển hộp danh sách thư mục (Dir List Box)
- Thuộc tính Pattern của điều khiển hộp danh sách tên tập tin (File List Box)

Tiếp đến sẽ trình bày cách thức trình ứng dụng FILESEL.VBP duy trì tính đồng bộ giữa các điều khiển này trong khi người dùng chọn ổ đĩa, thư mục và tập tin.

FILESEL.VBP QUẢN LÝ QUÁ TRÌNH CHỌN TẬP TIN

Khái niệm

Khi một điều khiển tập tin thay đổi, chẳng hạn điều khiển hộp danh sách ổ đĩa, các thay đổi luôn ảnh hưởng đến những điều khiển tập tin khác. Bạn phải biết cách viết mã lệnh các điều khiển để bắt tất cả chúng chỉ tới cùng ổ đĩa, thư mục và tập tin.

Khi người dùng nhấp nút lệnh Display Files, thủ tục biến cố Click trong Ví dụ 17.1 sẽ thi hành để cài đặt ổ đĩa và đường dẫn mặc định.

Ví dụ 17.1. *Mã lệnh khởi tạo sẽ hiển thị khung chọn tập tin.*

```
1: Sub cmdDisp_Click ()
2: ' Set default values
3: drvDrive.Drive = "c:"
4: dirList.Path = "c:\" ' Root directory
5:
6: ' Display the file-selection frame
7: lblDirs.Caption = dirList.Path
8: txtFiles.Text = "*.txt"
9: fraFile.Visible = True
10: ' Select the first file
11: ' in the list of files
12: filFile.ListIndex = 0
13: End Sub
```

Bằng cách định các thuộc tính Drive và Path trong hộp danh sách ổ đĩa và hộp danh sách thư mục thuộc dòng 3 và 4, nút lệnh sẽ khởi tạo một danh sách mặc định cho ổ đĩa C: và thư mục nguồn trên đĩa C:. Dòng 7 sẽ khởi tạo nhãn chỉ tên thư mục được chọn ở trên hộp danh sách thư mục (ngay dưới nhãn Directories:). Dòng 8 sẽ khởi tạo hộp nhập File Name với mẫu chọn tập tin mặc định. Phần mở rộng tên tập tin thường dùng cho các tập tin văn bản là txt, cho nên mẫu chọn ban đầu được đặt là *.txt.

Dòng 9 thực hiện công việc hiển thị khung đến người dùng. Trong suốt quá trình thiết kế FILESEL.VBP, thuộc tính Visible của điều khiển khung (Frame) được đặt là False, cho nên người dùng không thấy khung hay các điều khiển của nó cho đến khi người dùng nhấn nút lệnh. Cuối cùng, dòng 12 sẽ chọn mục đầu tiên trong hộp danh sách tập tin để một tên tập tin luôn được chọn trong danh sách và hiển thị trong khung. Nếu không có tập tin nào tồn tại với ổ đĩa, thư mục và mẫu chọn, dòng 12 không có tác dụng trong chương trình.

Mách nước

Trong quá trình quản lý, ổ đĩa phải thay đổi đầu tiên.

Khi thiết kế khung chọn tập tin, hãy luôn viết lệnh cho thủ tục biến cố thay đổi ổ đĩa trước bất kỳ điều gì khác. Lúc người dùng thay đổi ổ đĩa, bạn phải đảm bảo rằng danh sách thư mục cũng sẽ thay đổi. Ví dụ 17.2 chứa thủ tục biến cố Change cho ổ đĩa.

Ví dụ 17.2. *Mã lệnh sẽ thi hành khi người dùng chọn một ổ đĩa khác.*

```
Sub drvDrive_Change ()
dirList.Path = drvDrive.Drive
End Sub
```

Thủ tục biến cố drvDrive_Change() không làm gì nhiều. Thực tế, mục đích của thủ tục là kích hoạt biến cố khác. Ngay khi người dùng thay đổi ổ đĩa, Ví dụ 17.2 sẽ làm việc để cập nhật đường dẫn thư mục tới thư mục của ổ đĩa mới (thư mục mặc định hiện hành được chọn trên ổ đĩa được chọn gần nhất). Vì vậy, khi người dùng chọn ổ đĩa mới, thủ tục trong Ví dụ 17.2 sẽ hoạt động để cập nhật danh sách thư mục, và sẽ kích hoạt thủ tục biến cố Change của thư mục trong Ví dụ 17.3.

Ví dụ 17.3. *Khi thư mục thay đổi, phải liệt kê lại các tập tin.*

```
Sub dirList_Change ()
' The selection of the directory
' list box changed
filFile.Path = dirList.Path
' Make sure that one file is selected
If (filFile.ListCount > 0) Then
' Select the first file
```

```
8: ' in the list of files
9: filFile.ListIndex = 0
10: End If
11:
12: lblDirs.Caption = dirList.Path
13: End Sub
```

Dòng 4 cập nhật hộp danh sách tập tin (File List Box) để phản ánh đường dẫn của thư mục mới được chọn. Các dòng 5 đến 10 đảm bảo một tập tin được chọn, ngay cả khi người dùng không chọn tập tin nào, dòng 9 sẽ gán chọn lựa tới tập tin đầu tiên trong danh sách. Dòng 12 cập nhật đề mục tên đường dẫn sẽ xuất hiện bên dưới nhãn Directories để chỉ thư mục hiện hành. Quá trình ràng buộc bây giờ được gắn với nhau để bất kỳ một thay đổi ổ đĩa nào cũng dẫn đến việc thay đổi danh sách thư mục và tập tin. Nếu người dùng chỉ thay đổi thư mục mà không thay đổi ổ đĩa, quá trình thay đổi trong thư mục (nhờ vào thủ tục biến cố dirList_Change()) cũng sẽ thay đổi.

Biến cố Change không được gắn với hành động thay đổi ổ đĩa – thư mục – tập tin, nhưng lại là biến cố quan trọng để duy trì tất cả hoạt động trong trình ứng dụng. Nếu người dùng muốn xem tập hợp các tập tin khác trong thư mục hiện hành, người dùng có thể nhập vào mẫu chọn tập tin mới trong hộp nhập File Name. Ngay khi người dùng nhập mẫu chọn mới, thủ tục biến cố Change được mô tả trong Ví dụ 17.4 sẽ thi hành.

Ví dụ 17.4. *Người dùng đã thay đổi mẫu chọn tên tập tin.*

```
Sub txtFiles_Change ()
filFile.Pattern = txtFiles.Text
End Sub
```

Bất kỳ thay đổi nào trong hộp nhập File Name đều lập tức dẫn đến quá trình thay đổi tương ứng trong danh sách các tập tin đã trình bày. Vì vậy, ngay khi người dùng thay đổi mẫu chọn mặc định *.txt thành *.exe, hộp danh sách tên tập tin sẽ cập nhật để chỉ trình bày các tập tin có phần mở rộng là .exe.

Người dùng được phép đóng hộp thoại chọn tập tin bằng cách dùng một trong ba phương thức sau:

- Nhấp nút lệnh OK
- Nhấp đúp một tên tập tin
- Nhấp nút lệnh Cancel

Ví dụ 17.5 trình bày thủ tục biến cố Click của nút lệnh OK. Khi người dùng đóng khung chọn tập tin, nhiều việc phải thực hiện. Tên tập tin hiện sáng cần được lưu giữ. Dòng 3 lưu giữ tên tập tin hiện sáng bằng cách đặt tên tập tin trong hộp nhập File Name với dòng 5, ngay cả khi không nhìn thấy khung (dẫn đến che tất cả điều khiển trên khung, kể cả hộp nhập). Hộp nhập vẫn có tác dụng với chương trình, và thủ tục con bất kỳ trong trình ứng dụng có thể truy xuất txtFiles.Text để xem liệu người dùng đã chọn tập tin nào. Thêm vào đó, các điều khiển ổ đĩa và thư mục vẫn giữ các giá trị được chọn của chúng, ngay cả khi người dùng không thể thấy các điều khiển này.

Ví dụ 17.5. *Nhấp nút OK để đóng thủ tục con.*

```
1: Sub cmdFileOK_Click ()
2: ' Save the file selected by the user
3: txtFiles.Text = filFile.List(filFile.ListIndex)
4: ' Hide the file selection frame
5: fraFile.Visible = False
6: ' Tell surrounding code that Cancel was not pressed
7: DidCancel = False ' User didn't press Cancel
8: ' Show the "You Selected" command button
9: cmdSel.Visible = True
10: End Sub
```

Có một biến module tên DidCancel, được định nghĩa trong thủ tục (General) của module, sẽ giữ giá trị True hoặc False, phụ thuộc vào việc người dùng nhấp nút Cancel hay không. Nếu Ví dụ 17.5 thi hành, người dùng đã chọn OK, chứ không phải Cancel, cho nên dòng 7 đặt biến DidCancel thành False để trong trường hợp ứng dụng bất kỳ sử dụng Frame sẽ biết nút Cancel được nhấn. Dòng 9 đóng thủ tục biến cố bằng cách ẩn khung chọn tập tin và tất cả điều khiển của nó.

Nếu người dùng nhấp đúp tên tập tin, thủ tục biến cố DblClick được trình bày trong Ví dụ 17.6 sẽ thi hành. Thủ tục biến cố DblClick

không làm gì hơn ngoài việc gọi thủ tục biến cố Click của nút lệnh OK. Do đó kết quả sau sẽ xảy ra: tập tin cho người dùng chọn trở thành tập tin được yêu cầu và khung sẽ biến mất khỏi tầm nhìn của người dùng.

Ví dụ 17.6. *Người dùng đã chọn một tập tin bằng cách nhấp đúp tên tập tin.*

```
Sub filFile_DblClick ()
' Double-clicking a selected filename
' does the same thing as selecting a
' file and pressing the OK button.
' Therefore, call the OK button's Click event.
Call cmdFileOK_Click
End Sub
```

Nếu người dùng nhấp nút lệnh Cancel, không có điều gì thay đổi trong chương trình ngoại trừ việc khung sẽ biến mất và biến DidCancel được ấn định thành True. Vì vậy, nếu bạn thêm khung này và các thủ tục biến cố của nó vào các trình ứng dụng riêng của bạn, biến DidCancel sẽ báo cho bạn biết người dùng đã nhấn Cancel. Ví dụ 17.7 sẽ trình bày thủ tục Click của nút lệnh Cancel.

Ví dụ 17.7. *Người dùng nhấp nút Cancel để đóng khung chọn tập tin.*

```
Sub cmdCancel_Click ()
' Used pressed the Cancel button
' Hide the frame and inform the program
DidCancel = True
fraFile.Visible = False
End Sub
```

Củng cố

Mã lệnh bên trong khung chọn tập tin sẽ mô tả cách các điều khiển tập tin liên lạc với những điều khiển khác. Thay đổi của người dùng đến một trong các điều khiển tập tin có thể tác động tới các điều khiển tập tin (File) khác. Hãy sử dụng mã lệnh điều khiển những thay đổi phụ này để quá trình thay đổi ổ đĩa hay thư mục sẽ cập nhật hộp danh sách tập tin thích hợp.

Xin nhớ rằng nếu bạn chấp nhận khung chọn tập tin này cho trình ứng dụng riêng của bạn, ổ đĩa sẽ xuất hiện ở đầu tên đường dẫn. Vì vậy, bạn không phải đọc hộp danh sách ổ đĩa như khi sử dụng đường dẫn đầy đủ được thuộc tính Path của hộp danh sách hỗ trợ.

Bài tập

Kiến thức tổng quát

1. Dữ liệu trong biến hay dữ liệu trong tập tin được xem là tồn tại lâu dài hơn?
2. Tại sao việc sử dụng một hộp thoại để lấy tên tập tin từ người dùng ít an toàn hơn việc sử dụng một hộp nhận dữ liệu (input box) để yêu cầu tên tập tin?
3. Hộp thoại là gì?
4. Bạn có thể xây dựng các hộp thoại như thế nào trong Visual Basic?
5. Cách bạn bắt chước quá trình hiển thị và che giấu các hộp thoại khi dùng những điều khiển khung (Frame)?
6. Có bao nhiêu điều khiển trên hộp công cụ làm việc với tập tin?
7. Bạn có thể định một hộp danh sách ổ đĩa mặc định khi nào: lúc thiết kế hay lúc chạy chương trình?
8. Thuộc tính điều khiển danh sách tập tin Pattern chứa những gì?
9. Tại sao bạn phải bảo trì đồng bộ các điều khiển tập tin của khung?
10. Khi một khung chứa các điều khiển tập tin, bạn nên quản lý điều khiển liên hệ với tập tin nào đầu tiên khi viết mã lệnh để giữ các điều khiển được đồng bộ?
11. Điều khiển tập tin nào nên thay đổi khi người dùng nhấp hộp danh sách ổ đĩa?
12. Điều khiển tập tin nào nên thay đổi khi người dùng nhấp hộp danh sách thư mục?
13. Cách một ứng dụng xác định người dùng đã nhấn nút Cancel của khung tập tin trong FILESEL.VBP?

14. Thủ tục biến cố nào sẽ giữ danh sách tập tin hiện hành khi người dùng thay đổi mẫu (pattern) của các tập tin được chọn?

15. Đúng hay Sai: Khi che giấu một khung, tất cả điều khiển trên khung đó cũng sẽ che giấu.

Viết mã lệnh...

16. Hãy viết một mẫu danh sách tập tin sẽ hiển thị tất cả tập tin có tên bắt đầu với ACCT.

17. Hãy viết một mẫu danh sách tập tin hiển thị tất cả tập tin có tên kết thúc với phần mở rộng ex1, ex2, ex3, và ...v.v....

18. Hãy viết mã lệnh đặt một danh sách ổ đĩa tên drvDisk để chỉ đến tất cả tập tin trên đĩa D: và đặt một danh sách thư mục tên direng chỉ tới tất cả tập tin trong thư mục VBPRIMER.

Phần nâng cao

Hãy thay đổi FILESEL.VBP để cho phép chọn các tập tin đã tồn tại hoặc nhập mới tên tập tin. Bạn sẽ phải thêm mã lệnh đóng khung để chương trình kiểm tra nội dung vào của hộp nhập so với danh sách tập tin được chọn.

Bài 18

Một vài thao tác nhập xuất tập tin đơn giản

- **Tạo tập tin trên đĩa**
- **Mở tập tin**
- **Đóng tập tin bằng lệnh Close**
- **Ghi tập tin bằng lệnh Write**
- **Đọc dữ liệu từ tập tin bằng lệnh Input#**
- **Quá trình đọc dữ liệu thực sự dễ dàng nhờ lệnh Line Input#**

Khái niệm mới

I/O nghĩa là dữ liệu nhập vào và xuất ra.

Bài này giải thích cách bạn có thể dùng mã lệnh Visual Basic để quản lý thao tác nhập xuất dữ liệu tập tin trên đĩa. Nếu bạn đã tập hợp dữ liệu từ người dùng và lưu chúng trong biến và mảng, bạn có thể cất dữ liệu vào đĩa để nạp lại sau đó. Hơn nữa, bạn truy xuất các tập tin trên đĩa từ trong Visual Basic cho thông tin mã hàng hóa, số lượng, công nợ khách hàng, và bất kỳ điều gì khác mà chương trình của bạn cần từ kho lưu trữ các tập tin dữ liệu định kỳ.

Khi làm chủ Visual Basic và nâng cấp các sản phẩm Visual Basic khác, bạn sẽ thêm các điều khiển bổ sung vào cửa sổ Toolbox. Có nhiều điều khiển truy xuất cơ sở dữ liệu qua đó bạn có thể thiết đặt cơ chế đọc và ghi dữ liệu trong cơ sở dữ liệu bằng cách dùng phần mềm Microsoft Access và Paradox. Dù cho các điều khiển này cung cấp nhiều chức năng mạnh mẽ và dễ dàng hơn, bạn vẫn có thể tự lập trình lấy, lúc đó bạn sẽ cần các quy tắc truy xuất đĩa. Bài này sẽ cung cấp một chút kiến thức về truy xuất đĩa, một số lệnh và hàm truy xuất đĩa quan trọng nhất cần để làm việc với các tập tin dữ liệu.

TẠO TẬP TIN TRÊN ĐĨA

Khái niệm

Tập tin là tập hợp dữ liệu liên quan đến các chương trình bạn mua và viết, các tài liệu của trình xử lý từ. Thông thường, bạn sử dụng Visual Basic để truy xuất tập tin dữ liệu và văn bản lưu trên đĩa.

Có nhiều loại tập tin trên đĩa máy tính. Mỗi tập tin được lưu dưới một tên duy nhất trong một thư mục và ổ đĩa. Vì vậy, không thể có hai hay nhiều tập tin có cùng tên trừ khi các tập tin được lưu trong thư mục hay ổ đĩa khác nhau.

Khái niệm mới

Tập tin dữ liệu lưu dữ liệu trên đĩa.

Bài này đề cập đến các tập tin dữ liệu. Tập tin dữ liệu có thể ở tất cả các dạng. Thông thường, những lập trình viên mới làm quen Visual Basic nên làm việc với các tập tin dữ liệu ở dạng văn bản tự nhiên. Tập tin văn bản có thể đọc bất kỳ loại chương trình nào, và bất kỳ chương trình nào cũng có thể đưa ra các tập tin văn bản. Đôi khi tập tin văn bản còn được gọi là tập tin ASCII bởi vì tập tin văn bản gồm các chuỗi ký tự ASCII.

Trước khi Visual Basic có thể truy xuất tập tin, bạn hoặc người dùng phải chỉ cho Visual Basic vị trí chính xác tập tin được lưu trên đĩa. Nếu người dùng của bạn đang chọn một tập tin, bạn muốn sử dụng khung chọn tập tin (được trình bày trong bài) để cung cấp cho người dùng khả năng thay đổi ổ đĩa, thư mục và tên tập tin một cách dễ dàng. Khi chương trình của bạn truy xuất một tập tin người dùng không biết, chương trình sẽ phải cung cấp ổ đĩa, thư mục và tên tập tin.

Ghi chú

Bài thực hành cuối chương là ứng dụng kết hợp khung hộp thoại tập tin mà bạn đã nắm bắt trong bài trước với các lệnh và hàm nhập xuất tập tin để thiết kế chương trình truy xuất và hiển thị tập tin đầy đủ.

Củng cố

Bài này chỉ bạn cách truy xuất các tập tin dữ liệu văn bản được lưu trên đĩa. Bạn cần sự hỗ trợ của Visual Basic về tên tập tin, thư mục, và ổ đĩa cho tập tin bất kỳ mà Visual Basic làm việc.

MỞ TẬP TIN

Khái niệm

Lệnh Open sẽ mở các tập tin. Trước khi Visual Basic có thể truy xuất tập tin dữ liệu, trước tiên Visual Basic phải mở tập tin. Hãy nghĩ về lệnh Open đang làm việc trong Visual Basic như ngăn kéo chứa tập tin cần mở khi bạn muốn lấy một tập tin trong tủ. Lệnh Open tìm tập tin và chuẩn bị tập tin sẵn sàng cho Visual Basic.

Lệnh Open thực hiện các nhiệm vụ khác nhau như tìm tập tin, bảo đảm tập tin tồn tại nếu cần, và tạo thư mục quản lý tập tin trong khi tập tin đang mở. Một chương trình Visual Basic luôn phải mở tập tin bằng lệnh Open, trước khi chương trình có thể đọc hoặc ghi dữ liệu tới tập tin đó. Sau đây là dạng thức lệnh Open:

```
Open FileNameStr [For mode] As [#]FileNumber
```

FileNameStr phải là một chuỗi hay biến lưu tên tập tin. Tên tập tin phải ở trên ổ đĩa hay thư mục hiện hành trừ khi bạn xác định đường dẫn đầy đủ cho tập tin này. Thông thường, bạn không truy xuất dễ dàng tới ổ đĩa và thư mục mặc định Windows hiện hành của người dùng, cho nên bạn luôn xác định ổ đĩa và tên đường dẫn đầy đủ trong đối số *FileNameStr* của lệnh Open.

Mode phải là một giá trị được đặt tên trong Bảng 18.1. Có các giá trị *mode* bổ sung, nhưng tập sách này sẽ không đề cập các giá trị *mode* riêng hay cao cấp. *Mode* báo cho Visual Basic biết chính xác những gì chương trình của bạn mong đợi thực hiện với tập tin một khi Visual Basic mở tập tin.

Bảng 18.1. *Các giá trị mode có thể có cho lệnh Open*

Mode	Diễn giải
Append	Báo cho Visual Basic biết rằng chương trình của bạn cần ghi đến cuối tập tin nếu tập tin đã tồn tại. Nếu tập tin chưa tồn tại, Visual Basic sẽ tạo tập tin để chương trình của bạn có thể ghi dữ liệu tới tập tin.

Input	Báo cho Visual Basic biết chương trình của bạn cần đọc tập tin. Nếu tập tin không tồn tại, Visual Basic sẽ đưa ra thông báo lỗi. Khi bạn sử dụng một khung chọn tập tin, Visual Basic sẽ không bao giờ đưa ra lỗi, bởi vì khung chọn tập tin bắt buộc người dùng chọn một tập tin hay hủy bỏ thao tác chọn.
Output	Báo cho Visual Basic biết chương trình cần ghi tới tập tin. Nếu tập tin không tồn tại, Visual Basic sẽ tạo tập tin. Nếu tập tin đã tồn tại, đầu tiên Visual Basic sẽ xóa tập tin đang tồn tại và tạo tập tin mới cùng tên.

Dấu # là tùy chọn, mặc dù đa số lập trình viên Visual Basic xác định dấu # không thuộc số thẻ tập tin (một số phiên bản Visual Basic trước vẫn yêu cầu dấu #). *FileNumber* xác định một thẻ từ 1 tới 255, liên kết với tập tin sẽ mở bằng thẻ đó. Sau khi mở tập tin thành công (giả sử không có lỗi như cửa sổ ổ đĩa đang mở), phần còn lại của chương trình sẽ sử dụng các lệnh và hàm nhập xuất để truy xuất tập tin. Số thẻ tập tin sẽ tồn tại với tập tin cho đến khi bạn sử dụng lệnh Close (xem phần tiếp theo) để giải phóng thẻ tập tin *FileNumber*. Sau khi giải phóng, số thẻ đó có thể sử dụng cho các tập tin khác.

Ghi chú

Như với tất cả mô tả tập tin DOS và Windows, bạn có thể xác định ổ đĩa, thư mục và tên tập tin bằng cách dùng chữ hoa hay chữ thường.

Bạn có thể đồng thời mở nhiều tập tin trong một chương trình. Mỗi lệnh truy xuất tập tin hướng trực tiếp hoạt động của nó đến tập tin xác định bằng cách dùng số hiệu thẻ tập tin *FileNumber* của nó.

Lệnh Open sau đây sẽ tạo và mở tập tin dữ liệu trên đĩa và gán cho tập tin số hiệu thẻ là 1:

```
Open "d:\data\myfile.dat" For Output As #1
```

Nếu đã biết rằng tập tin tồn tại và bạn cần thêm dữ liệu vào tập tin, bạn có thể sử dụng chế độ Append để thêm vào tập tin sau lệnh Open này:

```
Open "d:\data\myfile.dat" For Append As #1
```

Một chương trình Visual Basic có thể có nhiều tập tin mở cùng thời điểm. Có lệnh tùy chọn FILES trong tập tin CONFIG.SYS máy tính bạn dùng để xác định số tập tin tối đa có thể mở tại một thời điểm. Nếu

FileNumber là #1 đã sử dụng cho một tập tin khác được mở trước đó trong trình ứng dụng, bạn có thể gán tập tin sẽ mở một số khác như sau:

```
Open "d:\data\myfile.dat" For Append As #5
```

Khi đã sử dụng số thẻ tập tin *FileNumber* hiện hành, bạn không thể gán giá trị đó cho một tập tin khác.

Lệnh Open sau sẽ mở cùng tập tin cho quá trình đọc dữ liệu từ một chương trình khác:

```
Open "d:\data\myfile.dat" For Input As #2
```

Visual Basic hỗ trợ hàm có sẵn FreeFile() không nhận một đối số nào. FreeFile() sẽ trả lại giá trị số thẻ tập tin có hiệu lực kế tiếp. Ví dụ nếu bạn đã sử dụng thẻ #1 và #2 cho các tập tin đã mở, giá trị kế tiếp được trả lại từ FreeFile() sẽ là 3. FreeFile() là hàm có ích nhất khi viết các thủ tục con và hàm dùng chung để mở tập tin, các thủ tục có thể được gọi từ nhiều nơi trong trình ứng dụng. Mỗi vị trí gọi có số thẻ tập tin đang mở khác nhau. Thủ tục có thể lưu giá trị thẻ tập tin có hiệu lực kế tiếp như sau:

```
fNum = FreeFile()
```

Và sử dụng biến fNum lần lượt trong lệnh Open, các lệnh nhập xuất và lệnh Close. Nếu không xác định được có bao nhiêu tập tin đang mở, thủ tục luôn dùng thẻ tập tin kế tiếp trong dòng lệnh để mở tập tin của nó.

Củng cố

Lệnh Open liên kết với tập tin bằng cách sử dụng thẻ tập tin mà phần chương trình còn lại sẽ truy xuất tập tin với thẻ đó. Có ba giá trị *mode* xác định cách Visual Basic sử dụng tập tin. Nếu bạn muốn ghi vào một tập tin, không thể dùng chế độ Input, còn như muốn đọc tập tin, không thể dùng chế độ Output hay Append.

ĐÓNG TẬP TIN BẰNG LỆNH Close

Khái niệm

Lệnh Close sẽ thi hành công việc ngược với lệnh Open. Lệnh Close đóng tập tin bằng cách ghi dữ liệu bất kỳ cuối cùng vào tập tin, giải phóng tập tin cho các trình ứng dụng khác, và trả số thẻ tập tin

lại cho trình ứng dụng của bạn trong trường hợp bạn muốn dùng thẻ đó cho lệnh Open kế tiếp.

Cuối cùng, mỗi chương trình mở tập tin nên đóng các tập tin này. Lệnh Close sẽ đóng tập tin cho bạn. Có hai dạng lệnh Close:

```
Close [[#]FileNumber] [, ..., [#]FileNumber]
```

Và

```
Close
```

Dạng đầu tiên đóng một hay nhiều tập tin đang mở, bằng cách xác định tập tin qua số hiệu thẻ tập tin đang mở. Dấu # là tùy chọn trước một số hiệu thẻ tập tin bất kỳ. Dạng thứ hai của lệnh Close sẽ đóng tất cả tập tin đang mở hiện hành. Lệnh Close đóng bất kỳ tập tin nào đang mở mà không cần biết bạn đã mở tập tin với chế độ mode nào.

Mách nước

Nếu tạo tập tin bằng cách mở tập tin trong chế độ Output và sau đó đóng tập tin, bạn có thể mở lại tập tin đó trong cùng chương trình ở chế độ Input để đọc tập tin.

Lệnh sau đây sẽ đóng hai tập tin đang mở và có số thẻ là 1 và 3:

```
Close 1, 3
```

Lệnh sau sẽ đóng tất cả tập tin đang mở:

```
Close ' Closes ALL files
```

Củng cố

Sử dụng lệnh Close để đóng các tập tin đang mở trước khi chương trình của bạn kết thúc. Quá trình đóng tập tin cung cấp khả năng bảo toàn dữ liệu, cho nên cần đóng các tập tin đang mở sau khi bạn truy xuất những tập tin này. Nếu một lỗi xảy ra trong quá trình thi hành chương trình, các tập tin đã đóng sẽ được lưu an toàn trên đĩa, nhưng những tập tin đang mở lúc xảy ra lỗi có thể bị mất dữ liệu.

GHI TẬP TIN BẰNG LỆNH Write

Khái niệm

Lệnh Write# có lẽ dễ sử dụng hơn khi dùng để ghi dữ liệu vào một tập tin. Lệnh Write# ghi bất kỳ loại dữ liệu nào vào tập tin. Bằng

cách dùng các lệnh Input thích hợp, bạn có thể đọc dữ liệu đã ghi vào tập tin với lệnh Write#.

Lệnh Write# cho phép bạn ghi bất kỳ dạng dữ liệu nào vào tập tin được mở trên đĩa ở chế độ Output hay Append. Lệnh Write# ghi chuỗi, số, hằng, biến, hay tổ hợp các kiểu dữ liệu này vào một tập tin trên đĩa.

Sau đây là dạng thức lệnh Write#:

```
Write #FileNumber [, ExpressionList]
```

FileNumber phải là một thẻ tập tin được liên kết với một tập tin đã mở bằng chế độ Output. Nếu bạn không xác định các biến hay giá trị để ghi, lệnh Write# sẽ ghi một ký tự trở về đầu dòng và xuống dòng (ký tự ASCII 13 sau đó là ký tự ASCII 10) vào tập tin, đặt một dòng trống vào trong tập tin. Nếu bạn xác định nhiều giá trị danh sách biểu thức *ExpressionList*, Visual Basic sẽ ghi dữ liệu đó vào tập tin bằng cách sử dụng những qui ước sau:

- Lệnh Write# sẽ phân tách nhiều mục trên cùng dòng bằng cách thêm dấu phẩy vào giữa các giá trị.
- Lệnh Write# luôn luôn thêm ký tự về đầu dòng và xuống dòng vào cuối mỗi dòng được ghi.
- Lệnh Write# sẽ thêm dấu nháy kép vào các chuỗi trong tập tin. Các dấu nháy kép làm cho việc đọc chuỗi sau đó dễ dàng hơn.
- Lệnh Write# ghi các giá trị ngày giờ theo dạng sau:

  #yyyy-mm-dd hh:mm:ss#

- Lệnh Write# ghi giá trị #NULL# vào tập tin nếu không có dữ liệu (VarType bằng 1).
- Lệnh Write# không ghi gì cả khi dữ liệu trống (VarType bằng 0), nhưng cũng phân cách các giá trị trống bằng dấu phẩy nếu ghi nhiều giá trị trong một dòng đơn.

Mã lệnh trong Ví dụ 18.1 sẽ ghi bài thơ nổi tiếng đầu thế kỷ vào tập tin đĩa ODE.TXT. Nút lệnh cmdWrite sẽ kích hoạt quá trình thi hành mã lệnh với biến cố Click khi người dùng nhấp nút lệnh. Tập tin sẽ xuất hiện trên thư mục nguồn của ổ đĩa C:

Ví dụ 18.1. *Ghi bốn giá trị chuỗi vào tập tin ODE.TXT.*

```
1: Sub cmdWrite_Click ()
2: ' Creates a text file and
3: ' writes a poem to the file
4: Open "c:\ode.txt" For Output As #1
5:
6: ' Write the poem
7: Write #1, "Visual Basic, Visual Basic,"
8: Write #1, "oh how I long to see..."
9: Write #1, "A working application that"
10: Write #1, "means so much to me."
11:
12: ' Always close any open file
13: ' when done with the file
14: Close #1
15: End Sub
```

Khi Visual Basic đóng tập tin ở dòng 14, sẽ có tập tin trong thư mục nguồn của người dùng với nội dung sau:

```
"Visual Basic, Visual Basic,"
"oh how I long to see..."
"A working application that"
"means so much to me."
```

Thông thường, các dấu nháy kép không bao giờ xuất hiện khi bạn gán hay hiển thị các giá trị chuỗi. Dấu nháy kép thường dành cho lập trình viên xác định giá trị bắt đầu và kết thúc chuỗi dùng trong chương trình. Ngoại lệ duy nhất để hiển thị dấu nháy kép là lệnh Write# luôn thêm các dấu nháy kép quanh tất cả chuỗi dữ liệu, chứa trong biến hay hằng, mà lệnh Write# ghi vào tập tin.

Khái niệm mới

Append *nghĩa là thêm vào cuối một cái gì đó.*

Nếu mở tập tin với chế độ Append, lệnh Write# sẽ thêm dữ liệu vào cuối tập tin. Chương trình trong Ví dụ 18.2 sẽ thêm một dòng trống và đoạn thơ thứ hai trong bài thơ được lưu trong tập tin ODE.TXT.

Lưu ý

Hãy nhớ nếu bạn ghi vào một tập tin đã tồn tại bằng cách dùng giá trị chế độ Output, Visual Basic sẽ xóa nội dung nguồn và thay thế tập tin bằng dữ liệu kế tiếp.

Ví dụ 18.2. *Mã lệnh thêm dữ liệu vào cuối tập tin với lệnh Write#.*

```
Sub cmdAddIt_Click ()
' Adds to a text file already created
' by writing a second verse to the file
Open "c:\ode.txt" For Append As #1

' Add to the poem
Write #1, ' Writes one blank line
Write #1, "Visual Basic, to you I sing"
Write #1, "the songs a programmer knows..."
Write #1, "Listen carefully when I choose Run,"
Write #1, "or we'll surely come to blows."

Close #1

End Sub
```

Bạn sẽ có cơ hội dùng khăn tay trước khi xem bài thơ mới được mở rộng. Sau đây là kết quả với chế độ Append nếu bạn hiển thị nội dung tập tin ODE.TXT sau khi chạy thủ tục biến cố này:

```
"Visual Basic, Visual Basic,"
"oh how I long to see..."
"A working application that"
"means so much to me."
"Visual Basic, to you I sing"
"the songs a programmer knows..."
"Listen carefully when I choose Run,"
"or we'll surely come to blows."
```

Mách nước

Bạn có thể ghi dữ liệu từ các biến hay điều khiển trên mẫu biểu vào tập tin. Bất cứ nơi nào bạn có dữ liệu cần ghi, lệnh Write# của Visual Basic sẽ ghi dữ liệu đó vào tập tin đĩa bạn đã mở.

Tóm tắt

Ví dụ 18.3 trình bày thủ tục con nhận bốn mảng cho bốn kiểu dữ liệu khác nhau và ghi dữ liệu mảng đó vào tập tin VALUES.DAT trên đĩa. Lưu ý cách vòng lặp For đơn giản có thể sử dụng để ghi khối lượng dữ liệu lớn vào tập tin dữ liệu. Đối số thứ hai đã gửi tới thủ tục con dùng để chỉ tổng số thành phần mảng được định nghĩa.

Củng cố

Lệnh Write# là một trong những lệnh ghi dữ liệu vào tập tin dễ nhất trong ngôn ngữ Visual Basic. Lệnh Write# phân cách nhiều giá trị ghi vào bằng dấu phẩy và bao tất cả dữ liệu chuỗi trong các dấu nháy kép.

Ví dụ 18.3. *Mã lệnh ghi nội dung mảng vào tập tin với lệnh Write#.*

```
Sub WriteData (CNames() As String, CBalc() As Currency, CDate() As Variant, CRegion() As Integer)
' Writes array data to a file
Dim ctr As Integer ' For loop control
' Assumes that each array has the
' same number of elements defined
Dim MaxSub As Integer
MaxSub = UBound(CNames) ' The maximum subscript

' Write MaxSub lines to the file
' with four values on each line
Open "c:\mktg.dat" For Output As #1
For ctr = 1 To MaxSub
Write #1, CNames(ctr), CBalc(ctr), CDate(ctr), CRegion(ctr)
Next ctr
Close #1

End Sub
```

Kết quả

Sau đây là sáu dòng dữ liệu trong tập tin MKTG.DAT mà chương trình trong Ví dụ 18.3 có thể ghi:

```
"Adams, H", 123.41, #1997-11-18 11:34:21#, 6
"Enyart, B", 602.99, #21:40:01#, 4
"Powers, W", 12.17, #1996-02-09#, 7
"O'Rourke, P", 8.74, #1998-05-24 14:53:10#, 0
"Grady, Q", 154.75, #1997-10-30 17:23:59#, 6
"McConnell, I", 9502.32, #1996-07-12 08:00:03#, 9
```

Củng cố

Dòng 1 định nghĩa thủ tục con và nhận bốn mảng được truyền vào. Giả sử mỗi mảng phải cùng số chỉ số được định nghĩa. Mặc dù không phải tất cả mảng được truyền vào thủ tục có cùng số chỉ số được định nghĩa, nhưng ví dụ này đã giả sử như vậy. Dòng 6 và 7 sẽ định nghĩa và khởi tạo giá trị giữ giá trị chỉ số cực đại, cho nên vòng lặp For kế tiếp có thể ghi hết thảy giá trị mảng vào tập tin MKTG.DAT.

Vòng lặp For trong các dòng từ 12 đến 14 duyệt qua những thành phần mảng, ghi trên một dòng bốn giá trị mảng và thực hiện quá trình lặp. Visual Basic bao dữ liệu mảng chuỗi bằng các dấu ngoặc kép và bao dữ liệu ngày giờ Variant với các dấu #.

Các dấu # bao giá trị ngày giờ Variant giúp Visual Basic khi bạn đọc lại tuần tự các giá trị dữ liệu vào biến Variant. Như bạn có thể thấy, ngày có thể thiếu giờ hay giờ có thể thiếu ngày. Lệnh Write# vẫn ghi ngày giờ như thế trong giá trị variant.

Lệnh Close trên dòng 15 sẽ đóng tập tin và giải phóng số hiệu thẻ tập tin trả về cho chương trình.

ĐỌC DỮ LIỆU TỪ TẬP TIN BẰNG LỆNH Input#

Khái niệm

Lệnh Input# đọc dữ liệu từ tập tin và lưu dữ liệu tập tin trong các biến và điều khiển chương trình. Input# là lệnh đối ngược với lệnh Write#. Dùng lệnh Input# để đọc dữ liệu bất kỳ bạn ghi vào tập tin bằng lệnh Write#.

Lệnh Input# đọc dữ liệu vào danh sách các biến hay điều khiển. Sau đây là dạng thức lệnh Input#:

```
Input #FileNumber [, ExpressionList]
```

Mặt khác, lệnh Input# là ảnh phản chiếu của lệnh Write# đã ghi dữ liệu vào tập tin. Khi bạn viết chương trình cần sử dụng dữ liệu từ một tập tin, hãy tìm lệnh Write# của chương trình đã tạo tập tin dữ liệu nguồn đó và sử dụng cùng dạng thức cho lệnh Input#.

Mách nước

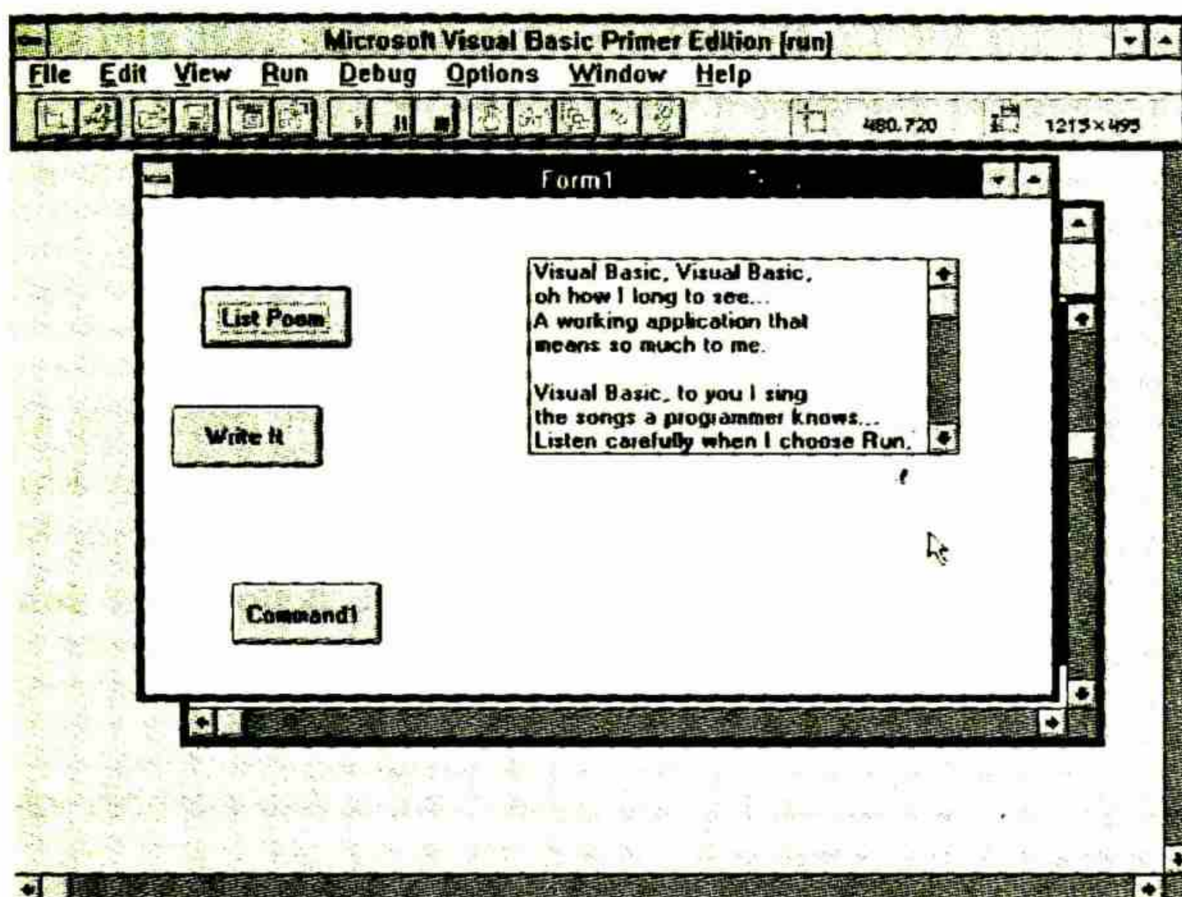
Phải mở các tập tin sẽ đọc bằng cách dùng giá trị chế độ mở tập tin Input, nếu không Visual Basic sẽ hiển thị thông báo lỗi.

Ví dụ 18.4 sẽ đọc và hiển thị quá trình cuộn bài thơ được lưu trong tập tin ODE.TXT vào hộp danh sách, tập tin ODE.TXT đã được tạo trong các ví dụ trước.

Ví dụ 18.4. *Mã lệnh đọc một bài thơ với lệnh Input#.*

```
Sub cmdList_Click ()
' Reads a poem from a text file and
' adds the poem, one line at a time,
' to the list box
Dim ctr As Integer ' For loop control
Dim PoemLine As String ' Holds each poem line
Open "c:\ode.txt" For Input As #1
' Read the poem
For ctr = 1 To 9
Input #1, PoemLine
lstPoem.AddItem PoemLine
Next ctr
' Always close any open file
' when done with the file
Close #1
End Sub
```

Hình 18.1 mô tả hộp danh sách sẽ xuất hiện như kết quả thi hành của Ví dụ 18.4.



Hình 18.1. *Nội dung tập tin trong hộp danh sách.*

Khái niệm mới

record ***là một hàng trong tập tin.***

Khi đọc dữ liệu từ tập tin, bạn rất dễ mắc lỗi vì cố tìm cách đọc nhiều dữ liệu hơn những dữ liệu tập tin đã lưu. Ví dụ 18.4 giả sử rằng chỉ có 9 mẩu tin trong tập tin bài thơ để đọc, và điều đó đã xảy ra trong trường hợp này. Tuy nhiên, với những tập tin dữ liệu lưu giá trị công nợ khách hàng hay tiền lương nhân viên, số mẩu tin sẽ thay đổi bởi vì bạn thường thêm bớt các mẩu tin khi diễn ra giao dịch.

Hàm Eof() là hàm kết thúc tập tin, có sẵn trong Visual Basic dùng để kiểm tra khi nào quá trình đọc dữ liệu gặp ký hiệu kết thúc tập tin. Sau đây là dạng thức hàm Eof():

```
Eof(FileNumber)
```

Eof() trả về giá trị True nếu quá trình đọc tập tin gần nhất của lệnh Input gặp ký hiệu kết thúc tập tin và trả về giá trị False nếu tập tin truy xuất vẫn còn dữ liệu để đọc. Đa số chương trình đọc dữ liệu lặp

cho đến khi điều kiện Eof() là True. Có lẽ cách tốt nhất để dùng Eof() là trong một vòng lặp Do Until–Loop có dạng tổng quát như sau:

```
Input #1, VariableList ' Read the first record
Do Until (Eof(FileNumer) = True)
 ' Process the record just read
 Input #1, VariableList ' Get more data
Loop
```

Nếu không có hoặc có một hay 400 mẩu tin trong tập tin, dạng Do Until này sẽ duy trì quá trình đọc, nhưng sẽ dừng ngay khi gặp ký hiệu kết thúc tập tin. Nhiều lập trình viên thường tăng một biến đếm nguyên trong vòng lặp để đếm số mẩu tin đọc được. Biến đếm có ích cho bạn nếu đang đọc dữ liệu tập tin vào mảng.

Ghi chú

Trường hợp bạn đọc dữ liệu tập tin vào mảng, hãy đảm bảo rằng mảng có số thành phần đủ hay nhiều hơn số mẩu tin tối đa cần lưu.

Tóm tắt

Ví dụ 18.5 đọc tập tin được ghi trước đây bằng cách dùng dãy lệnh Write# trong Ví dụ 18.3. Thân mã lệnh không được trình bày.

Củng cố

Ngay khi ghi dữ liệu vào tập tin bằng lệnh Write#, bạn sẽ dễ dàng đọc dữ liệu đó trở lại với lệnh Input#. Input# là lệnh ngược lại lệnh Write#. Kết hợp lệnh Write# và Input# làm cho quá trình nhập xuất tập tin trong Visual Basic đơn giản hơn so với đa số ngôn ngữ lập trình khác.

Ví dụ 18.5. *Quá trình đọc và báo cáo dữ liệu tập tin với lệnh Input#.*

```
Sub ReadData ()
' Reads array data from a file and reports the data
' Assume that 200 values were read
Static CNames(200) As String, CBalc(200) As Currency
Static CDate(200) As Variant, CRegion(200) As Integer
Dim NumVals As Integer ' Count of records
Dim ctr As Integer ' For loop control
```

```
9: NumVals = 1 ' Start the count
10: ' Reads the file records assuming
11: ' four values on each line
12: Open "c:\mktg.dat" For Input As #1
13: Input #1, CNames(NumVals), CBalc(NumVals), CDate(NumVals),
CRegion(NumVals)
14: Do Until (Eof(1) = True)
15: NumVals = NumVals + 1 ' Increment counter
16: Input #1, CNames(NumVals), CBalc(NumVals), CDate(NumVals),
CRegion(NumVals)
17: Loop
18:
19: ' When loop ends, NumVals holds one too many
20: NumVals = NumVals – 1
21:
22: ' The following loop is for reporting the data
23: For ctr = 1 To NumVals
24: ' Code goes here that outputs the array
25: ' data to the printer
26: '
27: Next ctr
28: Close #1
29:
30: End Sub
```

Phân tích

Dòng 4 và 5 định nghĩa 4 mảng lưu dữ liệu của tập tin. Các mảng được định nghĩa để lưu nhiều nhất 200 giá trị. Không có kiểm tra lỗi để đảm bảo quá trình đọc sẽ không đọc nhiều hơn 200 mẩu tin, nhưng bài tập cuối chương cho bạn cơ hội bổ sung quá trình kiểm tra mã lệnh cho mảng này.

Dòng 9 khởi tạo biến NumVals đếm số mẩu tin tập tin. Số đếm bắt đầu là 1 bởi vì chỉ số thành phần đầu tiên của mảng là 1. (Như với những chương trình trong sách, mã lệnh này bỏ qua chỉ số 0 mà các mảng sở hữu trừ khi bạn xác định Option Base 1.)

Dòng 12 sẽ mở tập tin để nhập dữ liệu vào và gán tập tin với số thẻ là 1. Lệnh Input đầu tiên ở dòng 13. Input sẽ thử đọc mẩu tin đầu tiên trong tập tin gồm bốn giá trị. Biểu thức kiểm tra quan hệ ở dòng 14 là điều kiện kết thúc tập tin sẽ trả về giá trị True ngay lập tức nếu tập tin rỗng. Ngược lại thân vòng lặp sẽ thi hành. Dòng 15 tăng số đếm mẩu tin thêm 1 để chỉ tới thành phần mảng sẽ nhận dữ liệu đọc với lệnh kế tiếp ở dòng 16. Vòng lặp sẽ tiếp tục cho đến khi kết thúc tập tin và biểu thức kiểm tra quan hệ của dòng 14 có giá trị False.

Dòng 20 phải giảm giá trị biến NumVals đi 1 bởi vì lần đọc tập tin cuối cùng luôn đem lại điều kiện kết thúc tập tin. Vì vậy, lệnh Input# cuối cùng không thành công và số đếm mẩu tin không thể tính cả lần đọc không thành công đó.

Sau khi đọc tất cả dữ liệu vào mảng, bạn có thể làm việc bình thường với dữ liệu mảng. Bạn có thể thống kê dữ liệu, hiển thị mảng bằng cách sử dụng các hộp danh sách và hộp combo, hay in dữ liệu ra giấy.

Quá trình đọc dữ liệu thực sự dễ dàng nhờ lệnh Line Input#

Khái niệm

Lệnh Line Input# đọc dữ liệu từ các tập tin dữ liệu đang mở. Không giống lệnh Input#, Line Input# đọc từng dòng dữ liệu trong tập tin vào một biến chuỗi. Bạn không phải xác định các tên biến riêng sau lệnh Line Input# bởi vì lệnh Line Input# đòi hỏi một giá trị chuỗi đơn. Lệnh Line Input# sẽ đọc dữ liệu từ tập tin bất kỳ có dòng kết thúc với ký tự xuống dòng và trở lại đầu dòng. (Đa số các tập tin đều như vậy.)

Đơn giản là lệnh Line Input# dùng để đọc các mẩu tin vào một biến đơn. Nếu lệnh Input# đọc từng giá trị mẩu tin riêng biệt, thì lệnh Line Input# lại đọc toàn bộ mẩu tin—dữ liệu, dấu phẩy, dấu ?, và các thứ khác—vào một biến chuỗi, biến Variant hoặc một điều khiển.

Sau đây là dạng thức lệnh Line Input#:

```
Line Input #FileNumber, VariableName
```

Không có cách đọc nhiều giá trị trong mẩu tin của tập tin có số thẻ tập tin là 3, câu lệnh Line Input# sau sẽ đọc một hình ảnh trong mẩu tin vào biến chuỗi aRecord:

```
Line Input #3, aRecord
```

Củng cố

Lệnh Line Input# đọc các mẩu tin trong tập tin dữ liệu vào biến chuỗi hay variant. Lệnh Input# đọc từng giá trị riêng lẻ của từng mẩu tin vào một biến hay điều khiển. Lệnh Line Input# sẽ đọc toàn bộ nội dung các mẩu tin vào biến chuỗi hay variant. Bài thực hành cuối chương sẽ sử dụng lệnh Line Input# để đọc nội dung tập tin vào một biến chuỗi đơn.

Bài tập

Kiến thức tổng quát

1. Tập tin là gì?
2. Mục đích của tập tin là gì?
3. Đúng hay Sai: Hệ thống máy tính của bạn có thể có nhiều tập tin có cùng tên.
4. Lệnh nào chuẩn bị một tập tin cho chương trình sử dụng?
5. Ba chế độ nào có thể có trong lệnh Open?
6. Mục đích của số tập tin là gì?
7. Nếu một tập tin không tồn tại và bạn mở tập tin để xuất, Visual Basic sẽ làm gì?
8. Nếu một tập tin đang tồn tại và bạn mở tập tin để xuất, Visual Basic sẽ làm gì?
9. Nếu một tập tin không tồn tại và bạn mở tập tin để thêm vào, Visual Basic sẽ làm gì?
10. Nếu một tập tin đang tồn tại và bạn mở tập tin để thêm vào, Visual Basic sẽ làm gì?
11. Nếu một tập tin không tồn tại và bạn mở tập tin để nhập, Visual Basic sẽ làm gì?
12. Vùng giá trị số hiệu tập tin của lệnh Open là gì?
13. Đúng hay Sai: Khi mở một tập tin, bạn phải xác định tên tập tin bằng cách dùng chữ thường.
14. Lệnh CONFIG.SYS nào giới hạn số tập tin mà bạn có thể mở tại một thời điểm?
15. Hàm nào tìm số hiệu tập tin có tác dụng kế tiếp?
16. Lệnh nào xóa tập tin sau khi hoàn thành lệnh với tập tin đó?
17. Tại sao giới thiệu lệnh Close?

18. Lệnh nào sẽ đóng tất cả các tập tin đang mở?
19. Lệnh nào ghi dữ liệu vào tập tin?
20. Lệnh Write# dùng ký tự nào để phân chia các giá trị xuất ra?
21. Ký tự nào lệnh Write# đặt trước và sau giá trị ngày giờ?
22. Ký tự nào lệnh Write# đặt trước và sau chuỗi?
23. Lệnh nào là ảnh phản chiếu của lệnh ghi dữ liệu Write#?
24. *Record* là gì?
25. Hàm nào kiểm tra kết thúc tập tin?
26. Lệnh nào sẽ đọc nội dung một mẩu tin vào một biến chuỗi hay Variant?

Kết quả xuất ra là gì?

27. Nội dung record là gì sau khi thực hiện lệnh Write# sau?

```
Write #1, "Peach", 34.54, 1, "98"
```

Tìm lỗi kỹ thuật

28. An Huy muốn đọc nội dung một mẩu tin vào biến chuỗi tên MyRecord, nhưng lệnh sau đây đã không làm việc. Hãy giúp An Huy giải quyết vấn đề.

```
Input #1, MyRecord.
```

Lập trình...

29. Viết lệnh Close đóng tập tin ở chế độ Input tương ứng với thẻ số 3 và tập tin ở chế độ Output tương ứng với thẻ số 19.
30. Hãy viết lệnh Open cần để mở tập tin tên NAMES.DAT trong chế độ Append. Hãy gán tập tin với số thẻ là 7.
31. Hãy thay đổi Ví dụ 18.5 để hiển thị một hộp thông báo và thoát vòng lặp đọc tập tin nếu tập tin dữ liệu nhập vào có nhiều hơn 200 mẩu tin. (Mảng lưu dữ liệu mới nhập không thể chứa hơn 200 giá trị.)

Phần nâng cao

Hãy viết một thủ tục con ghi bốn mẩu tin về bốn người bạn thân nhất của bạn gồm tên, tuổi, cân nặng và chỉ số IQ (hãy nhớ đây là bạn cho nên phải lịch sự). Viết thủ tục thứ hai đọc dữ liệu đó vào các biến chuỗi, mỗi lần một mẩu tin.

Bài thực hành 9

Mở rộng dữ liệu và chương trình

Tóm tắt

Chương này chỉ cho bạn cách tạo khung chọn tập tin giống hệt hộp thoại tập tin với các điều khiển tập tin trên hộp công cụ. Bạn đã biết rằng mã lệnh phải duy trì tính đồng bộ giữa các điều khiển tập tin với nhau, nếu không sẽ xảy ra nhiều vấn đề, chẳng hạn danh sách thư mục chỉ tới ổ đĩa không chứa thư mục nào trong hộp danh sách thư mục được liệt kê.

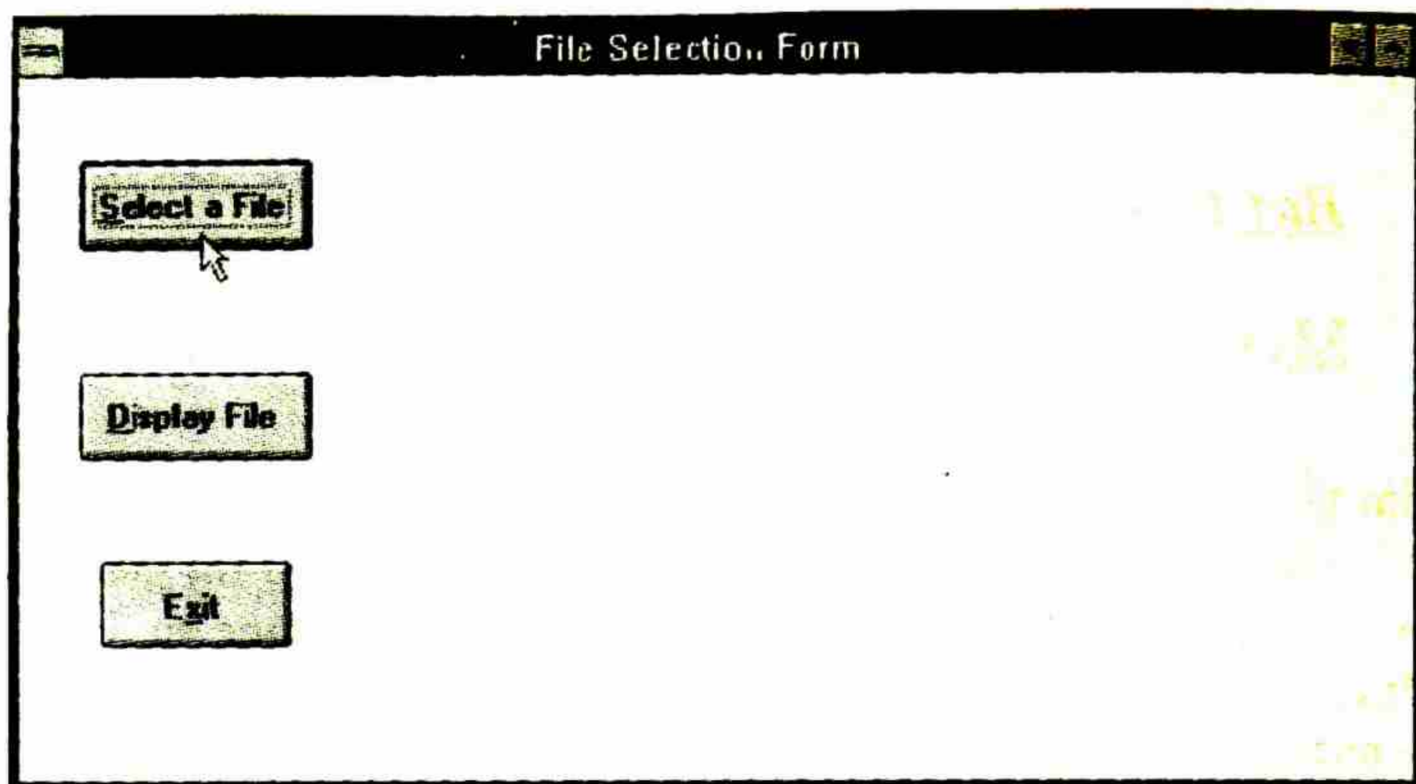
Bạn đã nắm bắt cách sử dụng các lệnh và hàm nhập / xuất tập tin trong Visual Basic để đọc, thêm và ghi tập tin dữ liệu. Quá trình truy xuất các tập tin dữ liệu khá đơn giản.

Trong chương này, bạn đã tìm hiểu:

- Lý do cần hộp thoại tập tin cho quá trình chọn tập tin
- Cách thao tác các điều khiển tập tin để duy trì tính đồng bộ của chúng
- Cách mở các tập tin và liên kết tập tin với một giá trị thẻ tập tin
- Lý do lệnh Write# là lệnh hoàn hảo cho quá trình gửi dữ liệu đến một tập tin dữ liệu ở dạng dễ đọc
- Khi nào dùng lệnh Input# và khi nào dùng lệnh Line Input# để đọc dữ liệu từ các tập tin

Mô tả chương trình

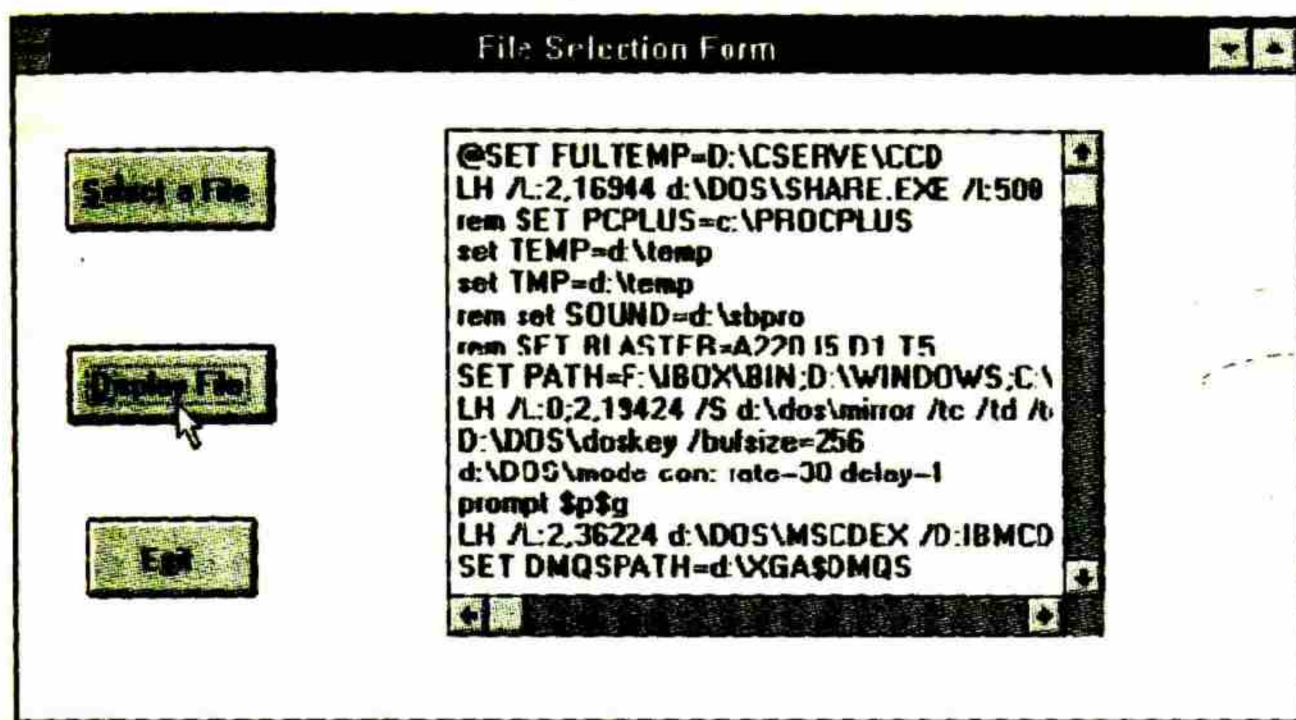
Hình P9.1 minh họa cửa sổ Form đang mở của PROJECT9.VBP. Bài thực hành này vô cùng đơn giản với nhiệm vụ hiển thị ba nút lệnh.



Hình P9.1. *Trình ứng dụng Project9 khởi đầu bằng một mẫu biểu.*

Hai điều khiển, khung chọn tập tin và hộp nhập lớn trên mẫu biểu sẽ không được hiển thị khi người dùng vận hành chương trình ban đầu. Khi người dùng chọn nút lệnh Select a File, khung chọn tập tin đã học trong Bài 17 sẽ xuất hiện. Chương trình chờ đợi người dùng chọn một tên tập tin.

Ngay khi người dùng chọn tên tập tin bằng cách nhấp đúp vào tên, hoặc nhấn nút lệnh OK trên khung chọn tập tin, khung chọn tập tin sẽ biến mất (thuộc tính Visible của nó được định là False) và mẫu biểu chỉ còn lại ba nút lệnh đơn giản. Sau đó nếu người dùng nhấp nút lệnh Display File, chương trình sẽ đọc nội dung của tập tin được chọn vào một biến chuỗi (single hoặc long). Tiếp đến chương trình sẽ hiển thị biến chuỗi đó trong hộp nhập mà thuộc tính Visible của nó được đổi thành True để người dùng có thể thấy và cuộn xem nội dung tập tin. Hình P9.2 minh họa tập tin mẫu được hiển thị trong hộp nhập.



Hình P9.2. *Hiển thị nội dung tập tin được chọn.*

Thực hành quá trình đọc tập tin

Phần chính của PROJECT9.VBP gồm điều khiển khung (Frame) và mã lệnh được trình bày trong điều khiển khung chọn tập tin ở Bài 17. Điểm khác biệt chính nằm ở mã lệnh Click của nút lệnh Display File trong PROJECT9.VBP.

Lưu ý

Nếu bạn hiển thị tập tin không phải là tập tin văn bản ASCII, tập tin sẽ hiển thị dữ liệu không chính xác trong hộp nhập. Chẳng hạn, nếu chọn bảng tính Microsoft Excel, bạn sẽ không thấy bảng tính trong hộp nhập, mà sẽ thấy trình bày mã nhị phân được nén của bảng tính.

Mã lệnh sử dụng lệnh Line Input# để đọc từng mẩu tin trong tập tin người dùng đã chọn trong khung chọn tập tin. Phần mô tả sẽ giải thích mã lệnh trong thủ tục cmdDisp_Click() trình bày ở Ví dụ P9.1.

Ví dụ P9.1. *Đọc toàn bộ tập tin vào biến chuỗi.*

```
Sub cmdDisp_Click ()
' Read the whole file (up to 30,000 characters)
' into a single string variable. Display that
' string variable in a text box.
Dim count As Integer ' Holds character count
```

```
Dim FileSpec, ALine As String
Dim FileHold As String ' Holds entire file
Dim NL As String
NL = Chr$(13) + Chr$(10)
' Gather the filename into a single string
' Add an ending backslash if the path has none
If (Right$(dirList.Path, 1) <> "\") Then
FileSpec = dirList.Path & "\" & txtFiles.Text
Else
FileSpec = dirList.Path & txtFiles.Text
End If
' Open the file
Open FileSpec For Input As #1
' Read up to the first 30,000 characters
Line Input #1, ALine ' Read a line
Do Until (EOF(1) = True)
ALine = ALine + NL ' Add a newline
' Make sure that the read won't overshoot string limit
If (count + Len(ALine)) > 30000 Then
Exit Do
End If
FileHold = FileHold & ALine
count = Len(FileHold) ' Update count
Line Input #1, ALine ' Read a line
Loop
Close #1
txtFile.Text = RTrim$(FileHold)
txtFile.Visible = True
End Sub
```

Mô tả

1: Nút lệnh hiển thị tập tin tên cmdDisp, do đó tên chương trình con thủ tục biến cố là cmdDisp_Click().

2: Chú thích giúp giải thích mục đích thủ tục.

3: Tiếp tục chú thích trước.

4: Tiếp tục chú thích trước.

5: Định nghĩa một biến nguyên lưu chiều dài biến chuỗi khi chương trình đọc tập tin vào biến.

6: Định nghĩa biến chuỗi tên FileSpec lưu tên đường dẫn và tên tập tin. Định nghĩa biến chuỗi ALine lưu nội dung từng mẩu tin đọc từ tập tin.

7: Định nghĩa một biến chuỗi lưu toàn bộ nội dung tập tin.

8: Định nghĩa một biến chuỗi lưu ký tự trở lại đầu dòng và xuống dòng mới.

9: Dòng trống giúp phân tách các phần thủ tục.

10: Định nghĩa ký tự bắt đầu dòng mới bằng cách nối ký tự trở lại đầu dòng và xuống dòng với nhau.

(10: Sử dụng bảng ASCII khi bạn cần nối với mã lệnh điều khiển.)

11: Dòng trống giúp phân tách các phần mã lệnh.

12: Chú thích giải thích phần thủ tục này.

13: Tiếp tục chú thích dòng trước.

14: Nếu đường dẫn được chọn không kết thúc với ký tự \, mã lệnh phải thêm nó.

(14: Ký tự \ luôn ở trước tên tập tin.)

15: Nối ký tự \ sau đường dẫn được chọn và trước tên tập tin.

16: Ngược lại... (đường dẫn được chọn đã kết thúc với ký tự \).

17: Nối đường dẫn được chọn với tên tập tin.

18: Kết thúc lệnh If.

19: Dòng trống giúp phân tách từng phần mã lệnh.

20: Chú thích giải thích mục đích phần mã lệnh này.

21: Mở tập tin được chọn để đọc. Gán tập tin với số thẻ File bằng 1.

22: Dòng trống giúp phân tách từng phần mã lệnh.

23: Chú thích giúp giải thích phần mã lệnh này.

24: Đọc mẩu tin đầu tiên.

25: Vòng lặp kết thúc tập tin chưa được gặp.

26: Thêm ký tự bắt đầu dòng mới vào mẩu tin vừa đọc.

(26: Lệnh Line Input# không đọc tổ hợp ký tự bắt đầu dòng mới của tập tin.)

27: Chú thích giải thích phần mã lệnh này.

28: Biến chuỗi có thể chỉ chứa ít hơn 30.000 ký tự. Vì vậy, bỏ qua mẩu tin vừa đọc và thoát khỏi quá trình đọc nếu chiều dài mẩu tin vượt quá 30.000 ký tự.

29: Kết thúc thủ tục nếu đạt tới giới hạn chuỗi.

30: Kết thúc lệnh If.

31: Dòng trống giúp phân tách từng phần mã lệnh.

32: Thêm mẩu tin vào biến chuỗi lưu tất cả mẩu tin đọc đến điểm đó.

33: Cập nhật tổng số chiều dài tập tin mới.

34: Đọc dòng tiếp theo từ tập tin.

35: Tiếp tục vòng lặp.

36: Đóng tất cả tập tin khi đã hoàn thành.

37: Dòng trống phân tách từng phần mã lệnh.

38: Cắt các khoảng trống bất kỳ vượt quá giới hạn khỏi chuỗi tập tin và hiển thị nội dung tập tin trong hộp nhập.

39: Hiện hộp nhập.

40: Kết thúc thủ tục Subroutine.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Chương kế tiếp sẽ giải thích cách thêm menu vào trình ứng dụng và thao tác điều khiển mới, điều khiển Timer.

Chương X

Bài 19

Menu

- **Cửa sổ Menu Design**
- **Thêm thanh menu**
- **Bổ sung menu xổ xuống**
- **Nối lệnh menu với thủ tục biến cố**
- **Thêm chức năng trợ giúp**

Bài này giải thích cách thêm menu vào trình ứng dụng. Tại sao phải thêm menu? Như bạn đã biết, menu bổ sung các lệnh cho trình ứng dụng mà nút lệnh không thể xử lý được. Mặc dù nút lệnh thường thực hiện các nhiệm vụ như những tùy chọn menu, nhưng người dùng Windows vẫn muốn thấy menu. Các lệnh chương trình chung như thoát chương trình thường ở trên danh sách lệnh của menu.

Làm thế nào xác định những lệnh sẽ có trên menu? Hãy nhìn một số chương trình Windows phổ biến. Đa số chương trình Windows có các lệnh menu và những đặc tính chung. Visual Basic là một chương trình như thế. Nhiều menu xổ xuống trong Visual Basic có chứa các lệnh giống như Microsoft Word và Microsoft Excel.

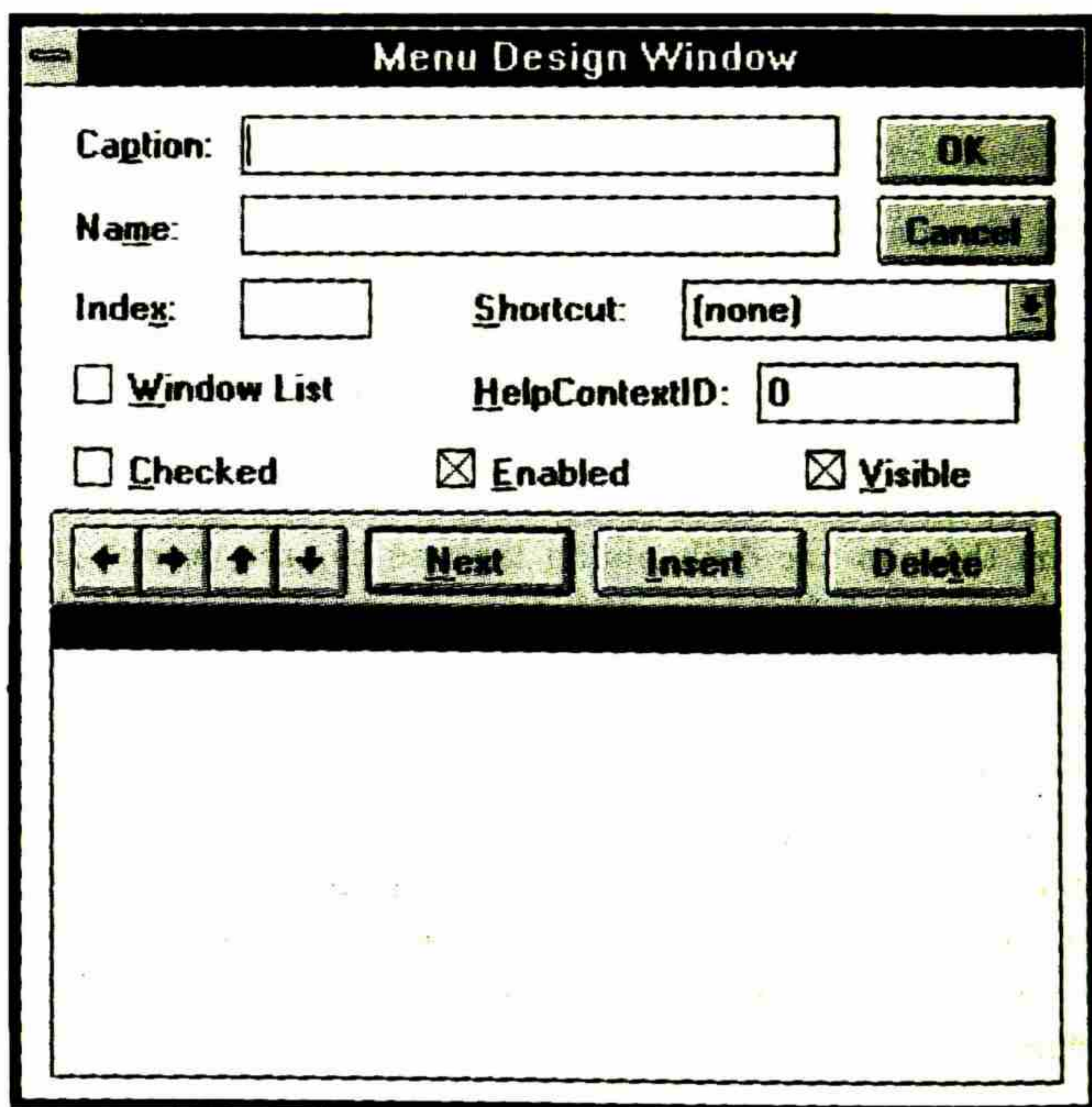
Những sản phẩm Microsoft không chỉ là chương trình Windows với lệnh menu chung. Khi viết một chương trình, bạn muốn người dùng cảm thấy thoải mái với giao diện. Người dùng sẽ sử dụng chương trình của bạn mà không cảm thấy phiền toái chỉ khi giao diện của bạn có các lệnh chung.

CỬA SỔ MENU DESIGN

Khái niệm

Visual Basic thực hiện quá trình tạo lập và thiết đặt các lệnh trên thanh menu vào trình ứng dụng cũng dễ dàng như đặt nút lệnh và nhập vào phím truy xuất nhanh. Cửa sổ Menu Design có các công cụ mô tả menu cho phép tạo thanh menu, lệnh menu, và phím truy xuất nhanh cho trình ứng dụng.

Cửa sổ Menu Design là một hộp thoại. Bạn có thể truy xuất từ cửa sổ Form bằng cách nhấn tổ hợp phím Ctrl+M hay chọn lệnh Window ➥ Menu Design từ thanh menu riêng của Visual Basic. Hình 19.1 trình bày cửa sổ Menu Design và tên các thành phần trong cửa sổ Menu Design.



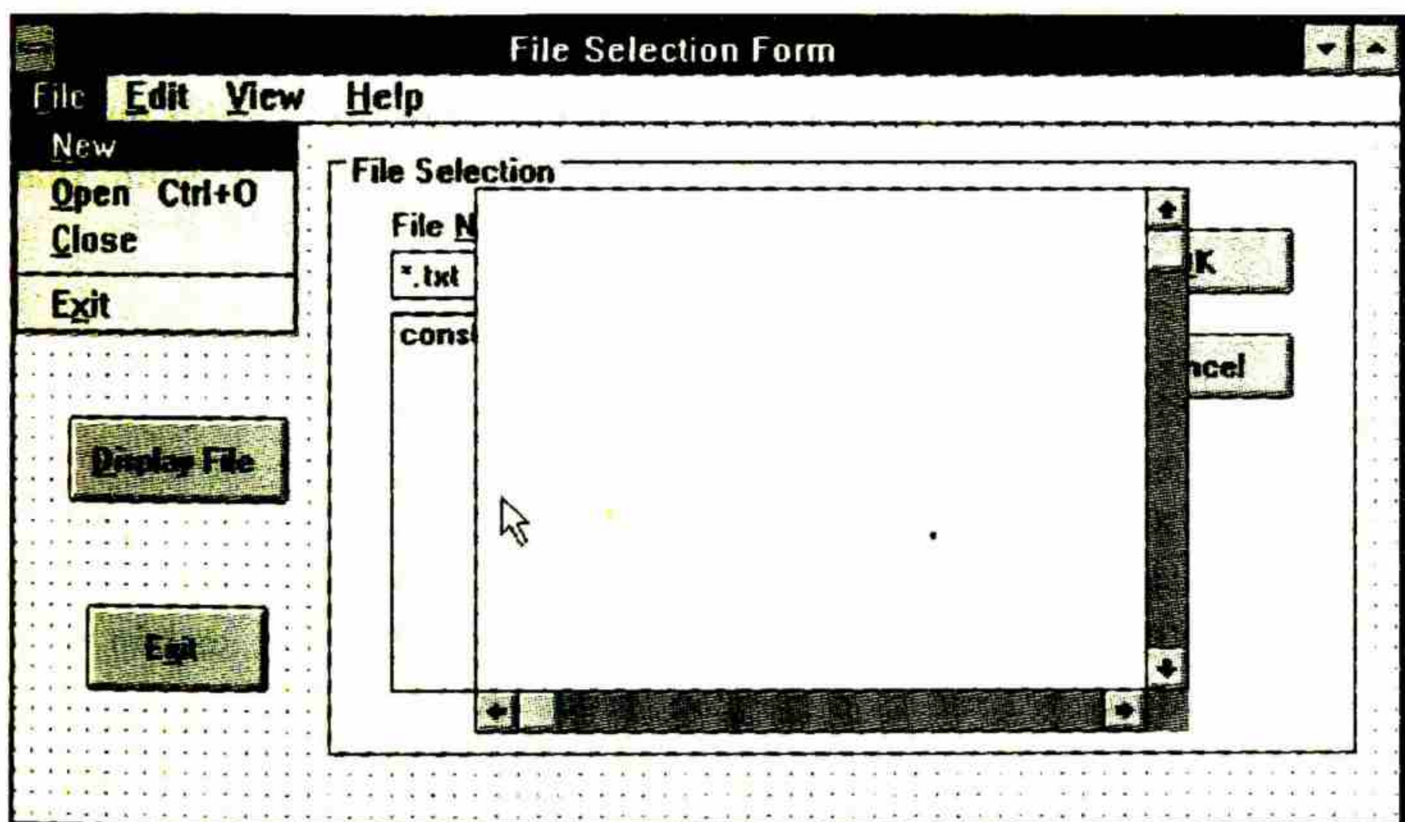
Hình 19.1. *Cửa sổ Menu Design.*

Cửa sổ Menu Design tạo menu cho bạn, nhưng bạn vẫn cần viết các thủ tục biến cố để gắn vào những tùy chọn lệnh menu sao cho trình ứng dụng có thể hoạt động. Khi người dùng chọn một lệnh menu, Visual Basic sẽ phát sinh một biến cố, như khi người dùng nhấp nút lệnh vậy.

Lưu ý

Để thêm menu vào chương trình, bạn phải làm chủ cửa sổ Menu Design. Sau khi sử dụng cửa sổ Menu Design tạo menu, các thủ tục biến cố của menu sẽ làm việc như những thủ tục biến cố khác đã trình bày trong sách.

Mở đầu quá trình bổ sung thanh menu vào trình ứng dụng, hãy xem Hình 19.2 để biết tên về những thành phần liên quan đến một thanh menu. Bài này sẽ hướng dẫn bạn tạo menu bằng cách thêm một thanh menu vào trình ứng dụng tên PROJECT9.VBP ở bài trước.



Hình 19.2. *Các thành phần trong menu.*

Lưu ý

Bài này không thêm thủ tục biến cố cho từng lệnh menu trong PROJECT9.VBP mà chỉ giải thích cách tạo menu và các thành phần của menu, cũng như giải thích cách móc các thủ tục biến cố vào menu. Một khi hiểu cách tạo và thêm thủ tục biến cố vào một vài lệnh menu, hẳn nhiên bạn sẽ biết cách thêm các thủ tục biến cố vào tất cả lệnh menu còn lại.

Củng cố

Thêm menu vào trình ứng dụng bằng cách dùng cửa sổ Menu Design. Cửa sổ này cho phép bạn thêm thanh menu, lệnh menu xổ xuống, đường phân cách, và tổ hợp phím truy xuất nhanh vào lệnh menu. Sau khi tạo menu, bạn sẽ viết các thủ tục biến cố cho từng lệnh menu. Khi người dùng chọn một lệnh menu, thủ tục biến cố của lệnh menu đó sẽ tự động thi hành.

THÊM THANH MENU

Khái niệm

Thanh menu của trình ứng dụng là phần dễ thêm nhất trong hệ thống menu. Phần này sẽ hướng dẫn bạn các bước cần thiết cho quá trình thêm một thanh menu. Phần kế tiếp sẽ thêm các mục menu xổ xuống cho từng lệnh trong thanh menu.

Cửa sổ Menu Design giúp bạn dễ dàng thêm một thanh menu vào bất kỳ ứng dụng nào. Hãy nạp tập tin PROJECT9.VBP và thêm thanh menu có các lệnh sau :

- File
- Edit
- View
- Help

Trước khi tiến hành, hãy cất các tập tin và mẫu biểu PROJECT9.VBP dưới một tên khác. Bằng cách này bạn sẽ bảo vệ nội dung project đúng với mô tả ở cuối bài trước. Dĩ nhiên, nếu tình cờ thay đổi PROJECT9.VBP, bạn có thể sao chép các tập tin PROJECT9.FRM và PROJECT9.VBP từ nơi lưu trữ vào lại đĩa cứng của bạn.

Các bước sau giả sử rằng bạn đã nạp tập tin PROJECT9.VBP và muốn cất bản sao tập tin mẫu biểu và project dưới cái tên MYMENU.FRM và MYMENU.VBP.

1. Chọn File ➥ Save File As mở hộp thoại Save File As và nhập vào tên mới, MYMENU.FRM, tại dấu nhắc File Name. Nhấn phím Enter hay nhấp OK đóng hộp thoại. Hộp thoại File Save As chỉ cất tập tin mẫu biểu.

2. Chọn File ➥ Save Project As mở hộp thoại Save Project As và nhập vào tên mới MYMENU.VBP tại dấu nhắc File Name. Nhấn phím Enter hay nhấp OK đóng hộp thoại.

Bây giờ bạn đã có bản sao PROJECT9.VBP là MYMENU.VBP. Bạn có thể sửa đổi bằng cách thêm menu. Bài thực hành của Chương 9 sẽ không bị đụng đến.

Mỗi lệnh trên thanh menu, cũng như các lệnh menu và đường phân cách sẽ xuất hiện khi bạn hiển thị menu xổ xuống, có những thuộc tính như các điều khiển khác. Cửa sổ Menu Design hoạt động giống như một hộp thoại giúp bạn thiết đặt các giá trị thuộc tính menu. Cửa sổ Property đầy đủ thuộc tính như bao điều khiển khác, nhưng bạn sẽ thấy, menu đòi hỏi thêm một vài loại thuộc tính bổ sung mà các điều khiển khác không cần đến.

Với trình ứng dụng MYMENU.VBP đã nạp, hãy thực hiện các bước sau để thêm thanh menu:

1. Nhấn tổ hợp phím Ctrl+M mở cửa sổ Menu Design. Trong phần này, bạn chỉ thêm các lệnh trên thanh menu. Mỗi lệnh đòi hỏi một đề mục (được xác định bởi thuộc tính Caption) và một tên (được xác định bởi thuộc tính Name).

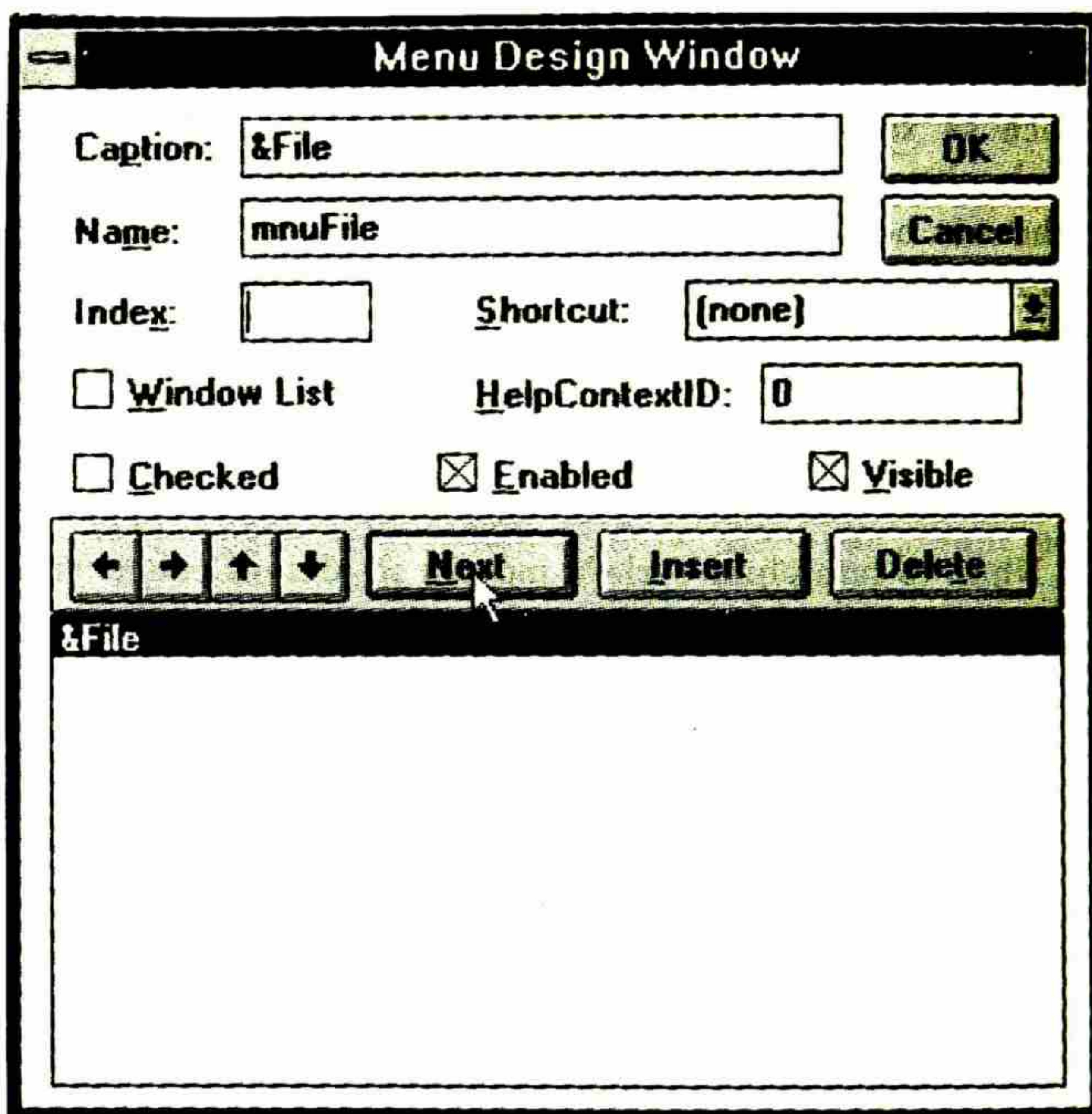
   Không bắt buộc, nhưng bạn có thể thiết đặt các thuộc tính khác như thuộc tính Enabled, để xác định mục menu được tô xám và không có hiệu lực với thủ tục xác định, hay thuộc tính Visible để xác định khi nào người dùng có thể nhìn thấy lệnh trên thanh menu. Thông thường, bạn ít khi thay đổi giá trị mặc định của các thuộc tính bổ sung này trên thanh menu.

2. Tại dấu nhắc Caption, hãy nhập **&File**. Dấu & trước đề mục File chỉ tổ hợp phím truy xuất nhanh là Alt+F cho mục menu File. Khi bạn nhập đề mục, chú ý rằng Visual Basic sẽ thêm đề mục ở phần dưới cửa sổ Menu Design.

Lưu ý

Ở nửa dưới cửa sổ Menu Design chứa nội dung mô tả menu đầy đủ, ngược lại nửa trên cửa sổ Menu Design chỉ chứa nội dung mô tả cho một mục menu riêng biệt trong menu.

3. Nhấn Tab di chuyển tiêu điểm đến hộp nhập Name, và nhập **mnuFile**. Trình ứng dụng sẽ hiểu mục thanh menu File tên là mnuFile. Các xác lập thuộc tính còn lại trong cửa sổ Menu Design chẳng có gì đáng nói. Màn hình của bạn sẽ được trình bày như Hình 19.3.



Hình 19.3. *Lệnh File trên thanh menu đã được thêm vào menu.*

Phím truy xuất nhanh có hiệu lực với lệnh tương ứng trên thanh menu khi chọn tổ hợp phím Alt+phím có dấu gạch dưới tương ứng với ký tự gạch dưới trong thuộc tính Caption. Đừng thử chọn tổ hợp phím Ctrl+phím nhấn từ hộp danh sách xổ xuống Shortcut cho các lệnh trên thanh menu. Những tổ hợp phím truy xuất nhanh Ctrl+phím nhấn chỉ có hiệu lực cho các lệnh menu xổ xuống.

Đừng nhấn phím Enter hay nhấp nút OK để đóng cửa sổ Menu Design, bởi vì bạn phải thêm các lệnh trên thanh menu trước khi đóng cửa sổ.

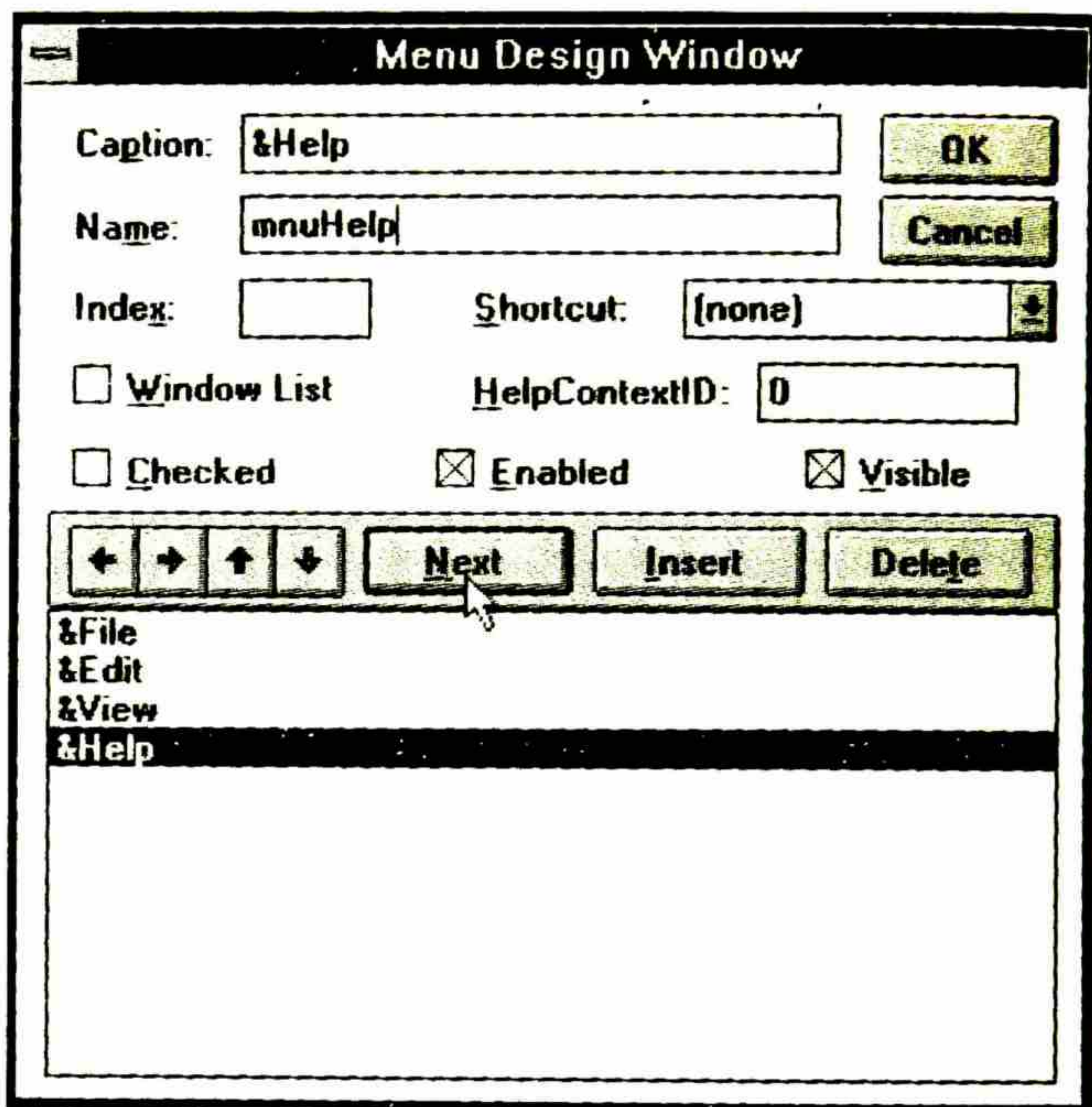
Tên các mục menu: Mã lệnh trình ứng dụng Visual Basic trong các thủ tục biến cố thường tham khảo các mục menu bằng tên mục menu đó. Hãy bắt đầu tên các mục menu trên thanh menu lẫn menu xổ xuống, bằng tiền tố đề mục mnu để bạn có thể dễ dàng phân biệt các lệnh menu với các biến và những điều khiển khác khi làm việc với mã lệnh chương trình.

Thông thường, lập trình viên Visual Basic tuân theo qui ước chuẩn đặt tên các mục menu sau tiền tố mnu. Vì vậy, lệnh File có tên là mnuFile, Edit có tên là mnuEdit, và ....

Khi bạn thêm các mục bổ sung vào menu xổ xuống, hãy đặt tiền tố mnu vào trước tên mục menu xổ xuống và tên lệnh trên thanh menu. Vì vậy, lệnh File ➡ Exit sẽ có tên là mnuFileExit, lệnh View ➡ Normal sẽ có tên là mnuViewNormal, và v.v. Các tên mô tả rõ ràng các mục menu.

Bổ sung lệnh trên thanh menu còn lại sẽ tốn ít thời gian hơn bằng cách thêm chúng vào phần dưới cửa sổ Menu Design. Các bước sau sẽ hoàn thành quá trình tạo thanh menu MYMENU.VBP:

1. Nhấp nút lệnh Next của cửa sổ Menu Design để báo cho Visual Basic biết bạn muốn thêm mục tiếp theo. Thanh sáng trong cửa sổ bên dưới sẽ nhảy tới dòng kế tiếp để chuẩn bị cho việc thêm mục menu kế tiếp.
2. Nhập **&Edit** vào hộp nhập Caption và nhấn Tab. Tên mục thứ hai trên thanh menu là **mnuEdit**. Nhấp nút lệnh Next để sửa soạn cửa sổ Menu Design cho mục kế tiếp trên thanh menu.
3. Nhập **&View** và nhấn Tab để chuyển tiêu điểm đến hộp nhập Name. Nhập **mnuView** và chọn Next để chuẩn bị cửa sổ Menu Design cho mục cuối cùng.
4. Nhập **&Help** và nhấn Tab để di chuyển tiêu điểm đến hộp nhập Name. Nhập **mnuHelp**. Màn hình của bạn bây giờ trông như Hình 19.4.



Hình 19.4. *Cửa sổ Menu Design với bốn mục menu được nhập.*

Đóng cửa sổ Menu Design bằng cách nhấn Enter hay nhấp nút lệnh OK. Ngay lập tức, Visual Basic sẽ hiển thị thanh menu mới ở trên cửa sổ Form của trình ứng dụng.

Củng cố

Cửa sổ Menu Design cung cấp những công cụ cần thiết để thêm thanh menu cùng với các lệnh và menu xổ xuống. Quá trình bổ sung các mục vào thanh menu là nhập tên và đầu đề của mục. Bây giờ bạn đã biết cách thêm các mục riêng biệt vào menu. Tiếp đến sẽ giải thích cách hoàn thành menu xổ xuống cho lệnh File trong thanh menu.

BỔ SUNG MENU XỔ XUỐNG

Khái niệm

Từng lệnh trên thanh menu sẽ mở một menu xổ xuống chứa một dãy lệnh, các đường phân cách và phím truy xuất. Menu xổ xuống đôi khi còn gọi là *menu con* (submenu). Bốn nút lệnh mũi tên bên trong cửa sổ Menu Design cho phép bạn phân cấp menu xổ xuống phù hợp với các lệnh trên thanh menu, để chỉ những mục này liên quan đến các lệnh nào trên thanh menu.

Bạn đã thêm thanh menu, bây giờ có thể bổ sung các mục riêng cho menu xổ xuống. Không nhất thiết phải nhập đầy đủ các lệnh trên thanh menu trước khi tạo từng menu xổ xuống. Bạn có thể thêm lệnh File vào thanh menu và sau đó tạo menu xổ xuống File trước khi thêm lệnh View vào thanh menu. Thứ tự thêm các mục menu không phải là tất cả. Menu xổ xuống File sẽ có các mục sau:

- Lệnh New
- Lệnh Open với tổ hợp phím truy xuất nhanh Ctrl+O
- Lệnh Close
- Đường phân cách
- Lệnh Exit

Sau khi thêm các mục menu con này, bạn có thể kết nối các lệnh menu với những thủ tục biến cố bạn viết.

Quá trình thêm các mục menu con y hệt quá trình thêm lệnh trên thanh menu. Điểm khác biệt là các tùy chọn cửa sổ Menu Design trở nên quan trọng hơn bởi vì bạn sẽ áp dụng những tùy chọn này thường xuyên hơn cho các mục menu xổ xuống. Bảng 19.1 sẽ giải thích các thuộc tính cửa sổ Menu Design còn lại.

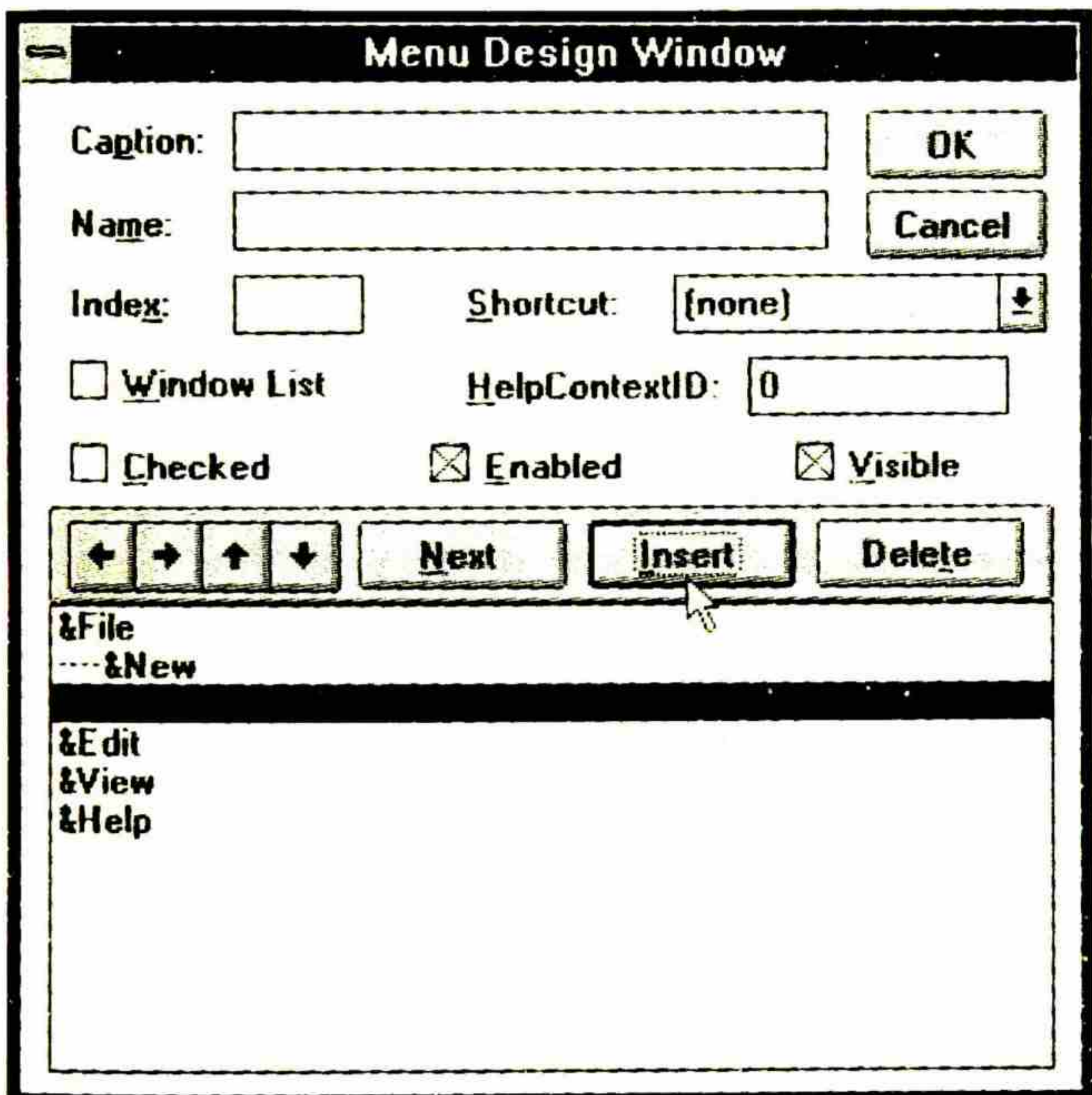
Bảng 19.1. *Giải thích thuộc tính còn lại của cửa sổ Menu Design*

Thuộc tính	Mô tả
Checked	Xác định liệu một mục menu có dấu kiểm bên cạnh mục hay không. Thông thường bạn sẽ thêm các dấu kiểm vào lệnh menu thi hành các hành động bật tắt, như là menu View chứa lệnh Highlighted. Dấu kiểm được bật trong lúc thiết kế hay thông qua mã lệnh bằng cách định thuộc tính Checked của lệnh menu là True. Dấu kiểm sẽ gỡ bỏ (cho biết lệnh không hoạt động hay không được chọn) khi bạn định thuộc tính Checked là False.
HelpContextID	Đây là mã lệnh tương ứng với mô tả tập tin trợ giúp xác định nếu thêm các tập tin trợ giúp vào trình ứng dụng.
Index	Nếu tạo mảng điều khiển menu thay cho tên các mục menu riêng biệt, thuộc tính Index xác định chỉ số mục menu trong mảng điều khiển.
Shortcut	Đây là danh sách xổ xuống tổ hợp phím truy xuất Ctrl+phím nhấn có thể thêm cho một lệnh bất kỳ trong menu xổ xuống.
Window List	Xác định lệnh menu áp dụng cho tài liệu MDI (giao diện đa tài liệu) của trình ứng dụng cao cấp. Menu bạn tạo trong sách không đòi hỏi sử dụng đặc tính MDI.

Có lẽ phím lệnh quan trọng nhất trên cửa sổ Menu Design khi thêm các mục menu xổ xuống là bốn nút lệnh mũi tên. Nút lệnh mũi tên trái và phải cho biết các lệnh này thuộc lệnh nào trong thanh menu. Nói cách khác, nếu bốn lệnh trong cửa sổ bên dưới được dịch sang phải và hiện ra ngay cạnh mục File trong thanh menu, bốn lệnh dịch phải này sẽ nằm trong menu xổ xuống File. Mũi tên trái sẽ giảm cấp lệnh menu và mũi tên phải sẽ tăng cấp lệnh menu lên. Mũi tên lên và xuống dùng để di chuyển các lệnh lên xuống trong danh sách lệnh menu, sắp xếp lại thứ tự nếu cần.

Các phím mũi tên rất có ý nghĩa khi bạn hiểu cách sử dụng chúng. Hãy thực hiện các bước sau để tạo menu con xổ xuống cho lệnh File trên thanh menu:

1. Di chuyển thanh sáng trong cửa sổ bên dưới tới mục **&Edit** của thanh menu. Nhấp nút lệnh Insert. Lệnh này luôn luôn chèn trước một mục, cho nên để thêm các mục vào menu File, bạn phải chèn trước mục Edit của thanh menu trong cửa sổ bên dưới.
2. Nhấp nút lệnh mũi tên phải. Visual Basic sẽ thêm một số khoảng trống, cho biết mục được chèn gần nhất sẽ là mục con dưới tùy chọn File.
3. Chuyển tiêu điểm đến dấu nhắc Caption và nhập **&New**.
4. Nhấn phím Tab để chuyển tiêu điểm đến dấu nhắc Name và nhập **mnuFileNew**.



Hình 19.5. *Quá trình thêm menu File.*

5. Nhấp Next và sau đó nhấp Insert để chèn mục khác ngay dưới mục New. Cửa sổ Menu Design sẽ trông giống như Hình 19.5.

6. Chuyển tiêu điểm đến dấu nhắc Caption và nhập **&Open**. Nhấn phím Tab và nhập **mnuFileOpen** vào dấu nhắc Name. Trước khi thêm mục tiếp theo, nhấp danh sách thả xuống Shortcut và chọn Ctrl+O. Khi người dùng hiển thị menu xổ xuống File, Ctrl+O sẽ xuất hiện như phím tắt bên cạnh mục menu File ➥ Open.

7. Nhấp Next, tiếp đó nhấp Insert để tạo không gian nhập mục tiếp theo. Thêm đề mục Close với tên mnuFileClose. Nhấp Next sau đó là Insert để chèn một mục khác ngay dưới mục Close. Bây giờ là lúc thêm đường phân cách.

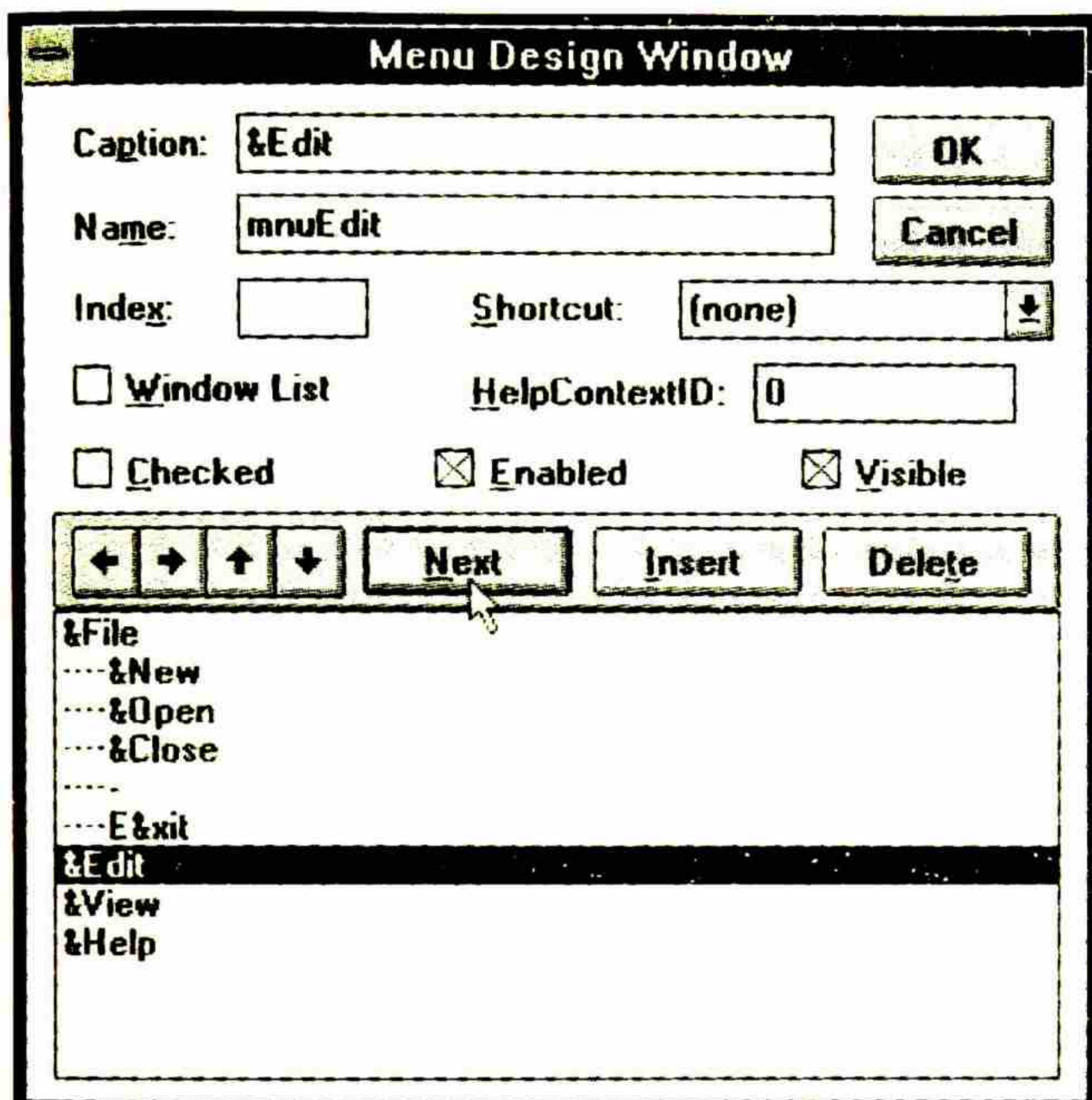
Các đường phân cách giúp bạn phân chia lệnh trong menu xổ xuống thành những phần riêng biệt. Mặc dù các lệnh này xuất hiện trên menu xổ xuống File của đa số trình ứng dụng Windows, nhưng không phải tất cả đều thực hiện cùng một chức năng. Một số lệnh liên quan đến tập tin, một số liên quan đến in ấn, và lệnh Exit luôn xuất hiện trên menu File. Các đường phân cách giúp phân biệt những nhóm lệnh khác nhau trong menu xổ xuống.

Tất cả đường phân cách có cùng thuộc tính Caption không có gì ngoài dấu nối, –. Bạn phải đặt tên khác nhau cho các đường phân cách. Thông thường, tên đường phân cách trên menu File là mnuFileBar1, mnuFileBar2, và .... Bạn phải thêm các đường phân cách bằng cách dùng nút lệnh mũi tên phải để chúng thụt vào đúng ngay dưới menu xổ xuống của chúng. Hãy thực hiện các bước sau để thêm đường phân cách cho menu xổ xuống File của MYMENU.VBP:

1. Nhập – cho vùng Caption và nhấn phím Tab.

2. Nhập **mnuFileBar1** cho vùng Name.

Còn một mục nữa cần thêm. Bạn đủ biết cách thêm lệnh Exit vào menu File. Sau khi thêm Exit, cửa sổ Menu Design sẽ trông giống như Hình 19.6.



Hình 19.6. *Các lệnh đầy đủ trong menu File.*

Một đặc tính phụ trong menu là thêm mục được đánh dấu kiểm vào menu xổ xuống View. Thêm mục con Highlighted với tên mnuViewHighlighted. Nhấp ô chọn Checked. Mục View Highlighted sẽ được đánh dấu kiểm khi người dùng chọn lệnh View Highlighted từ menu.

Bây giờ bạn đã hoàn thành menu, hãy nhấp nút lệnh OK. Khi cửa sổ Menu Design biến mất, bạn sẽ thấy cửa sổ Form của trình ứng dụng với thanh menu ở phía trên màn hình.

Củng cố

Quá trình thêm menu vào trình ứng dụng của bạn đòi hỏi bạn làm chủ cửa sổ Menu Design. Menu chính là điều khiển cao cấp với các giá trị thuộc tính bạn ấn định bằng cách sử dụng cửa sổ Menu Design. Đa số các mục menu đòi hỏi bạn xác định thuộc tính Caption và Name

cũng như canh dòng đúng những mục ngay dưới lệnh trên thanh menu của nó. Không bắt buộc, một mục menu có thể chứa tổ hợp phím truy xuất nhanh hay một dấu kiểm bên cạnh mục.

NỐI LỆNH MENU VỚI THỦ TỤC BIẾN CỐ

Khái niệm

Một khi đã thiết kế xong menu, bạn cần gắn từng lệnh menu vào trình ứng dụng của bạn. Để đáp ứng quá trình chọn menu, bạn phải viết thủ tục biến cố Click mà bạn muốn Visual Basic thi hành khi người dùng chọn một lệnh menu.

Visual Basic sẽ phát sinh biến cố Click khi người dùng chọn một lệnh menu. Tên lệnh menu kết hợp với từ Click, chính là tên thủ tục biến cố. Vì vậy, lệnh menu File ➡ Exit sẽ thi hành thủ tục biến cố mnuFileExit_Click().

Việc thêm thủ tục biến cố mnuFileExit_Click() chỉ yêu cầu bạn chọn lệnh menu đó trong quá trình thiết kế chương trình. Tại cửa sổ Form, hãy nhấp lệnh File trên thanh menu. Visual Basic sẽ hiển thị menu xổ xuống File. Thậm chí dù bạn không chạy chương trình nhưng đang làm việc trên chương trình từ cửa sổ Form, menu File sẽ xuất hiện để cho bạn biết điều gì sẽ xảy ra vào lúc người dùng chọn lệnh File khi chạy chương trình.

Nhấp lệnh Exit trên menu xổ xuống File. Ngay khi bạn nhấp lệnh Exit, Visual Basic sẽ mở cửa sổ Code với thủ tục biến cố mới tên mnuFileExit_Click(), như mô tả trong Hình 19.7.

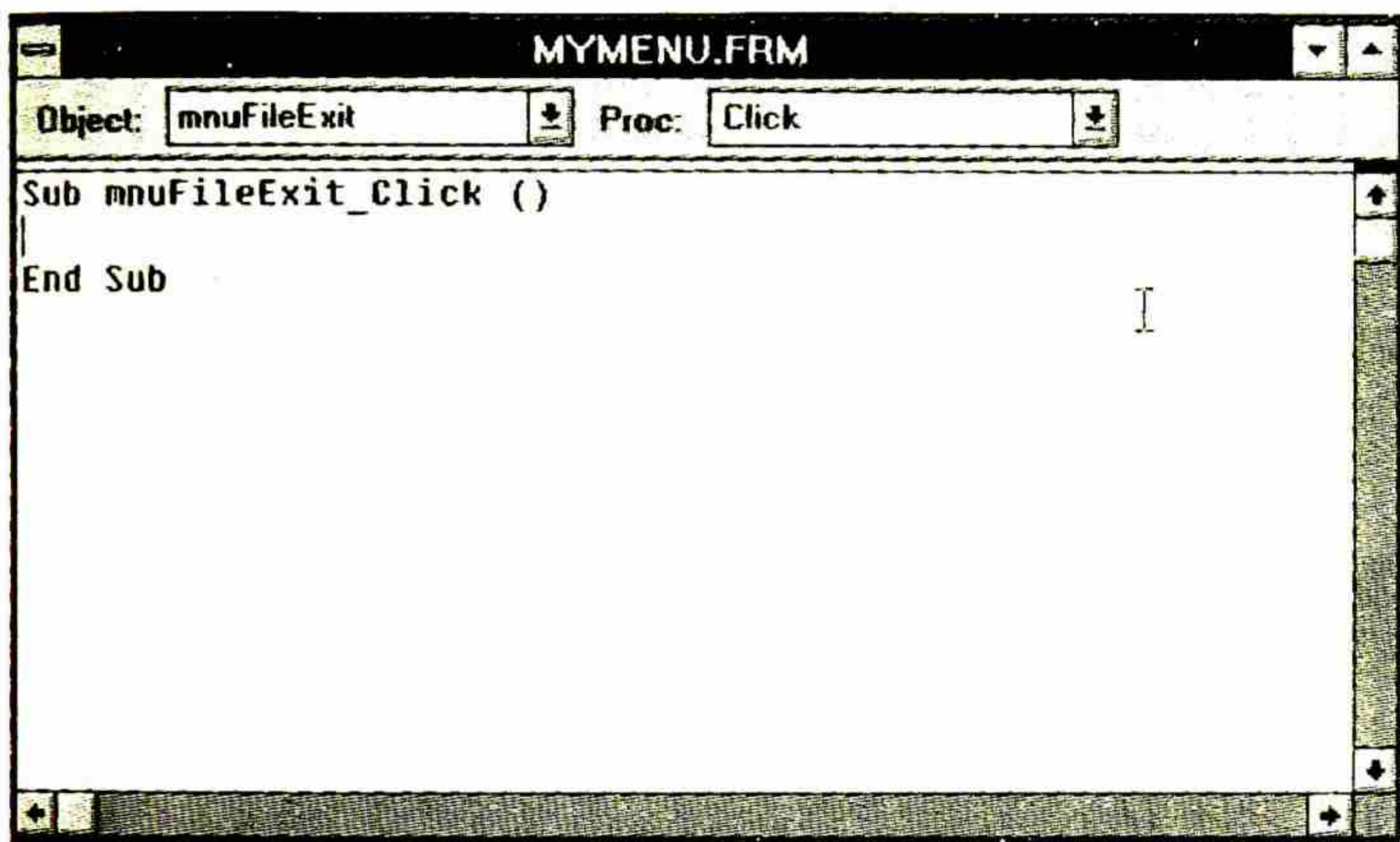
Thủ tục biến cố này khá đơn giản. Khi người dùng chọn File ➡ Exit, bạn muốn trình ứng dụng kết thúc. Vì vậy, hãy chèn lệnh End tới thân thủ tục biến cố mnuFileExit_Click() và đóng thủ tục bằng cách nhấp đúp nút điều khiển của nó.

Mặc dù bài này không dành thời gian hoàn thành tất cả menu xổ xuống cho các lệnh trên thanh menu và thêm mã lệnh thủ tục biến cố vào từng lệnh menu, nhưng nó đã chỉ cách kết nối lệnh menu File ➡ Open để hiển thị khung chọn tập tin khi người dùng chọn nút lệnh Display a File trên mẫu biểu. Dĩ nhiên bản sao nút lệnh Display a File ở bên trong thủ tục mnuFileOpen_Click(), bạn có thể thi hành biến cố

Click của nút lệnh bằng cách thêm dòng sau vào thân thủ tục mnuFileOpen_Click():

```
cmdSel_Click ' Execute the Display a file event
```

Như bạn thấy, quá trình thêm các thủ tục biến cố đòi hỏi một vài thao tác nhấp lệnh menu và mã lệnh được đưa vào thân thủ tục sẽ xuất hiện.



Hình 19.7. *Visual Basic mở cửa sổ Code khi bạn nhấp một mục menu.*

Tóm tắt

Hãy thêm thủ tục biến cố khác vào tùy chọn menu View Highlighted bằng cách nhấp lệnh menu của nó và thêm mã lệnh trong Ví dụ 19.1 vào thủ tục.

Củng cố

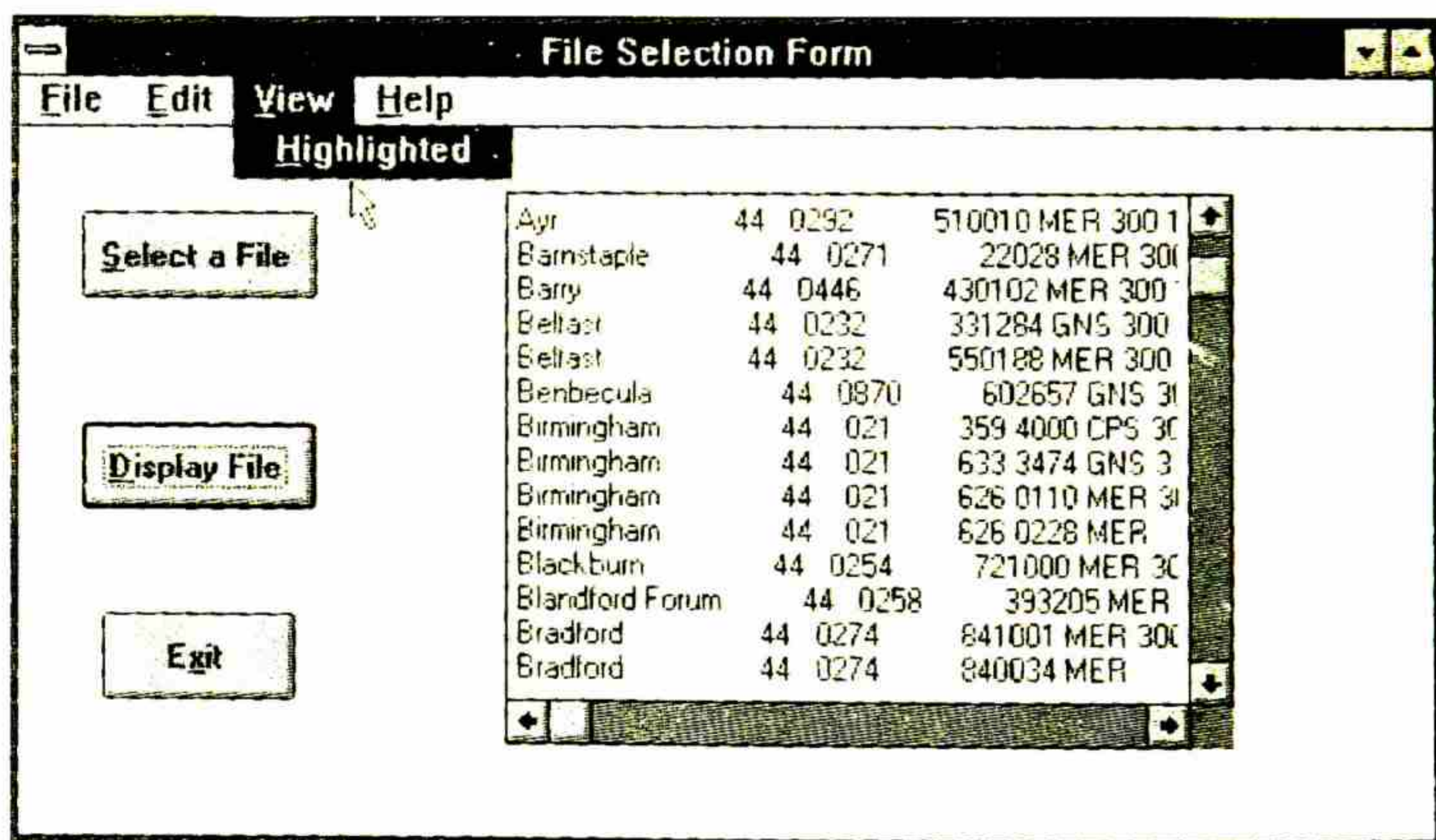
Sau khi thiết kế menu, bạn phải gắn mã lệnh với các mục menu khác nhau bằng cách viết thủ tục biến cố Click sẽ thi hành khi người dùng chạy trình ứng dụng và chọn menu. Nếu lệnh menu nào đó là bản sao chức năng của các thành phần điều khiển khác chẳng hạn như nút lệnh, đừng sao chép mã lệnh của nút lệnh này vào thân thủ tục biến cố menu. Thay vào đó, chỉ việc gọi thủ tục biến cố của nút lệnh đó ngay trong thủ tục biến cố của lệnh menu.

Ví dụ 19.1. *Mã lệnh in đậm tập tin được hiển thị .*

```
Sub mnuViewHighlighted_Click ()
' Determines if the file being displayed
' appears Boldfaced or not
mnuViewHighlighted.Checked = Not (mnuViewHighlighted.Checked)
txtFile.FontBold = mnuViewHighlighted.Checked
End Sub
```

Kết quả xuất ra

Hình 19.8 mô tả tập tin dữ liệu văn bản đang được hiển thị không có phông chữ đậm. Trước khi bạn chọn lệnh menu View ➡ Highlighted, tập tin luôn xuất hiện ở dạng chữ đậm.



Hình 19.8. *Khi không được chọn, lệnh menu View ➡ Highlighted sẽ tắt phông chữ đậm lúc hiển thị tên tập tin.*

Phân tích

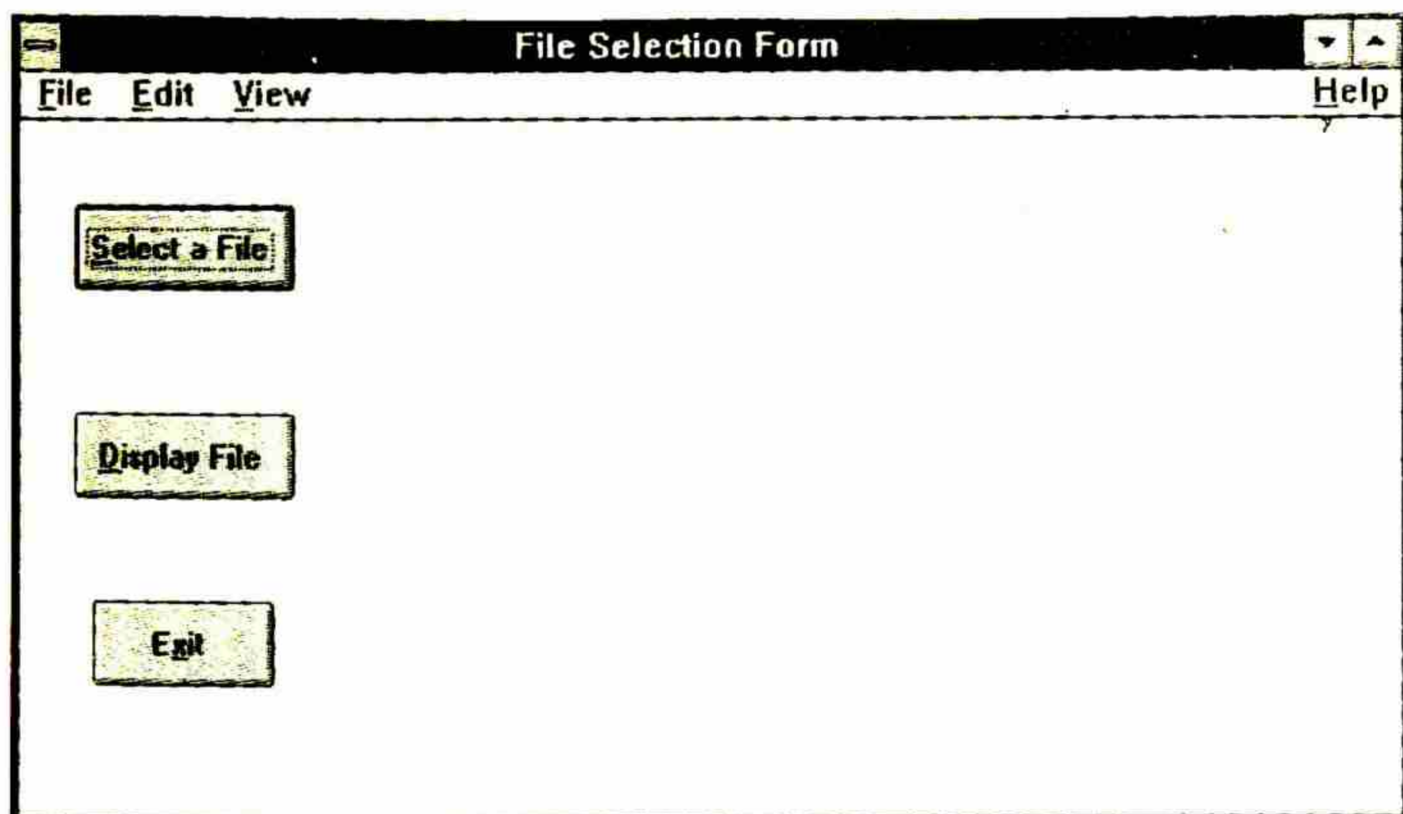
Đầu tiên, vào thời điểm người dùng chạy chương trình, lệnh View Highlighted sẽ được đánh dấu kiểm, nghĩa là tập tin mà người dùng hiển thị sẽ xuất hiện ở dạng chữ đậm.

Ví dụ 19.1 sử dụng toán tử Not để định thuộc tính FontBold của hộp nhập hiển thị tập tin trong trình ứng dụng. Dòng 4 định lại việc đánh dấu mục menu View Highlighted. Nếu mục menu được đánh dấu, dòng 4 sẽ thiết đặt giá trị ngược lại với toán tử Not. Dấu kiểm trên mục menu lúc đó sẽ biến mất và không được hiển thị khi người dùng hiển thị lại menu xổ xuống View. Dòng 5 ấn định thuộc tính FontBold của hộp nhập là TRUE hoặc FALSE ứng với tình trạng mục menu được đánh dấu hay không.

THÊM CHỨC NĂNG TRỢ GIÚP

Khái niệm

Sử dụng hàm Chr$(), bạn có thể thêm lệnh Help mà ít người biết đến vào bên phải thanh menu.



Hình 19.9. *Sử dụng một mẹo nhỏ bạn có thể thêm lệnh Help vào bên phải thanh menu.*

Một số trình ứng dụng sẽ hiển thị thanh menu giống như thanh menu bạn đã thêm trong MYMENU.VBP, ngoại trừ việc các trình ứng dụng này còn có lệnh Help xuất hiện ở góc phải thanh menu như Hình 19.9 minh họa.

Nếu thêm ký tự ASCII 8, ký tự điều khiển backspace, với thuộc tính Caption của lệnh thì lệnh sẽ xuất hiện ở bên phải thanh menu. Tại sao lại thêm ký tự điều khiển backspace? Không ai biết điều đó!

Bạn không thể thêm ký tự backspace trong quá trình thiết kế trình ứng dụng, nhưng lại có thể thêm ký tự backspace bằng cách dùng hàm Chr$() trong mã lệnh khởi động chương trình. Những gì bạn phải làm là nối đề mục của lệnh Help trên thanh menu với ký tự Chr$(8). Bạn phải nối Chr$(8) lúc chạy chương trình. Có lẽ vị trí tốt nhất để thêm Chr$(8) là trong thủ tục Form_Load() (thủ tục sẽ thi hành trước khi người dùng thấy mẫu biểu). Ví dụ 19.2 trình bày mã lệnh cần để canh phải tùy chọn Help trên thanh menu.

Ví dụ 19.2. *Mã lệnh canh phải lệnh Help trên thanh menu.*

```
1: Sub Form_Load ()
2: ' Right-justify the Help menu bar item
3: mnuHelp.Caption = Chr$(8) & mnuHelp.Caption
4: End Sub
```

Bài này không trình bày bất cứ điều gì với tùy chọn Help bởi vì quá trình thêm trợ giúp trực tuyến vào một trình ứng dụng nằm ngoài phạm vi cuốn sách này. Bạn có thể nối bất kỳ một lệnh trên thanh menu với Chr$(8) nếu muốn lệnh đó xuất hiện bên phải thanh menu.

Mách nước

Nếu nối ký tự Chr$(8) cho hơn một lệnh trên thanh menu, chỉ có lệnh sau cùng trên thanh menu tính từ trái qua mới xuất hiện ở góc phải thanh menu.

Củng cố

Bằng cách nối vào ký tự Chr$(8) lúc chạy chương trình, bạn có thể hiển thị lệnh ở góc phải thanh menu. Tuy nhiên, đừng lạm dụng khả năng này, bạn chỉ nên canh phải lệnh Help trên thanh menu.

Bài này kết thúc ở đây mà không thêm các thủ tục biến cố vào tất cả lệnh menu bởi vì bạn có thể thực hiện các bước tương tự như đã trình bày cho các lệnh còn lại. Bây giờ bạn đã hiểu cách thêm thanh menu, mã lệnh vào trình ứng dụng, và đã nắm rõ cách tạo các chương trình Windows chuyên nghiệp!

Bài tập

Kiến thức tổng quát

1. Công cụ Visual Basic dùng để thiết kế menu là gì?
2. Đúng hay Sai: Cửa sổ Menu Design cho phép bạn xác định các giá trị thuộc tính cho trình ứng dụng.
3. Đúng hay Sai: Bạn có thể thêm tổ hợp phím truy xuất nhanh Ctrl+phím nhấn vào các lệnh trên thanh menu.
4. Đúng hay Sai: Bạn có thể thêm tổ hợp phím truy xuất nhanh Alt+phím nhấn vào các lệnh trên thanh menu.
5. Tên tiền tố nên dùng cho tất cả mục menu là gì?
6. *Submenu* là gì?
7. Nút lệnh nào trong cửa sổ Menu Design cho phép bạn sắp xếp lại các tùy chọn menu?
8. Những nút lệnh nào trong cửa sổ Menu Design cho phép bạn phân cấp các mục menu cho biết những mục đó có phải là thành phần của một menu xổ xuống hay không?
9. Thuộc tính menu nào cho phép bạn thêm các dấu kiểm cạnh mục menu?
10. Thuộc tính menu nào cho phép bạn che giấu các mục menu vào những thời điểm khác nhau?
11. Đúng hay Sai: Các lệnh menu có thể là một phần của mảng điều khiển.
12. Đúng hay Sai: Thuộc tính Caption tạo các vạch phân cách trên menu bằng dấu nối (–).

13. Tên biến cố mà tất cả quá trình chọn lựa menu sẽ kích hoạt là gì?

14. Ký tự nào sẽ canh thẳng bên phải các mục menu?

15. Đúng hay Sai: Bạn có thể canh thẳng bên phải cho nhiều lệnh trên thanh menu.

Tìm lỗi kỹ thuật

16. An Huy muốn canh thẳng bên phải mục menu View: Anh ấy đã nhập vào thuộc tính Caption của mục menu này như sau :

```
Chr$(8) & &View
```

Hãy giải thích cho Frank cách thêm ký tự khoảng trống đúng.

Viết mã lệnh...

17. Bạn đặt tên cho một mục menu có phụ đề Split là gì khi người dùng chọn nó từ thanh menu Window?

18. Bạn sẽ đặt hai đường phân cách nào trên menu xổ xuống View?

19. Hãy mô tả cách bạn có thể tô xám (làm mất hiệu lực) các mục menu trong MYMENU.VBP nghĩa là không có mã lệnh thủ tục biến cố nào vận hành lúc đó (như File New chẳng hạn).

Phần nâng cao

Hãy tạo một trình ứng dụng trống với các mục thanh menu là A, B, C, D, và E. Viết mã lệnh trong thủ tục Form_Load() canh phải mục E. Trên từng menu xổ xuống, thêm các mục sau 1, 2, 3, 4, và 5. Chèn đường phân cách giữa mục 2 và 3, 4 và 5. Sau khi hoàn thành, hãy chạy chương trình để bảo đảm rằng tất cả các mục menu được trình bày như mong đợi. Xong xuôi bạn sẽ làm chủ các giai đoạn căn bản bổ sung các lệnh menu.

Bài 20

Định thời gian

- Giới thiệu về điều khiển bộ định thời
- Sử dụng điều khiển bộ định thời
- Các biến cố ít xảy ra
- Bộ định thời không phải là đồng hồ báo thức

Bài này mô tả điều khiển bộ định thời (Timer). Không giống đa số điều khiển khác, người dùng chẳng bao giờ nhìn thấy điều khiển bộ định thời, ngay cả khi điều khiển bộ định thời được đặt trên mẫu biểu như các điều khiển khác. Điều khiển bộ định thời đếm khoảng thời gian trôi qua và tự động kích hoạt các biến cố riêng của nó sau một khoảng thời gian định trước.

Khác với cách đáp ứng biến cố của các điều khiển khác, điều khiển bộ định thời sẽ tự kích hoạt và đáp ứng biến cố thông qua thủ tục biến cố bạn viết. Các xác lập thuộc tính của điều khiển bộ định thời xác định thời điểm phát sinh biến cố của điều khiển bộ định thời (Timer).

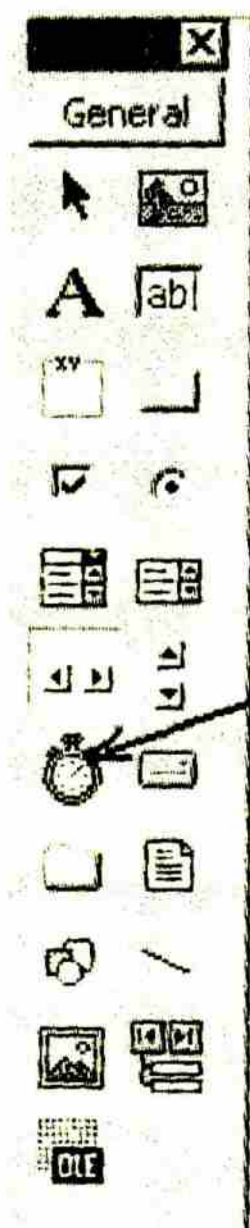
Ghi chú

Điều khiển bộ định thời không làm việc với hàm Timer() mà bạn đã học trong Bài 16. Bộ định thời là điều khiển xuất hiện trong cửa sổ Toolbox, còn Timer() là một hàm bạn thêm vào các chương trình bên trong cửa sổ Code.

GIỚI THIỆU VỀ ĐIỀU KHIỂN BỘ ĐỊNH THỜI

Khái niệm

Điều khiển bộ định thời ẩn khuất đâu đó trên mẫu biểu của người dùng, đếm từng khoảng thời gian xác định đã trôi qua.



Hình 20.1 cho thấy vị trí điều khiển bộ định thời xuất hiện trên hộp công cụ. Điều khiển bộ định thời giống như đồng hồ bấm giờ. Khi đặt điều khiển bộ định thời trên mẫu biểu, hình dạng điều khiển bộ định thời cũng giống như vậy. Hãy nhớ rằng người dùng không nhìn thấy điều khiển bộ định thời khi chạy trình ứng dụng.

Bạn có thể đặt điều khiển bộ định thời trên mẫu biểu tương tự như điều khiển khác. Bạn có thể thực hiện một trong hai cách sau:

- Nhấp đúp điều khiển bộ định thời để nó xuất hiện giữa mẫu biểu. Sau đó kéo điều khiển đến vị trí muốn đặt.
- Nhấp và kéo điều khiển tới vị trí đặt nó trên mẫu biểu.

Hãy đặt điều khiển bộ định thời ở ngoài các điều khiển khác. Luôn nhớ rằng người dùng không thể thấy điều khiển bộ định thời, cho nên bạn có thể đặt nó ở ngoài vùng các điều khiển để khỏi phiền hà điều khiển khác trên mẫu biểu.

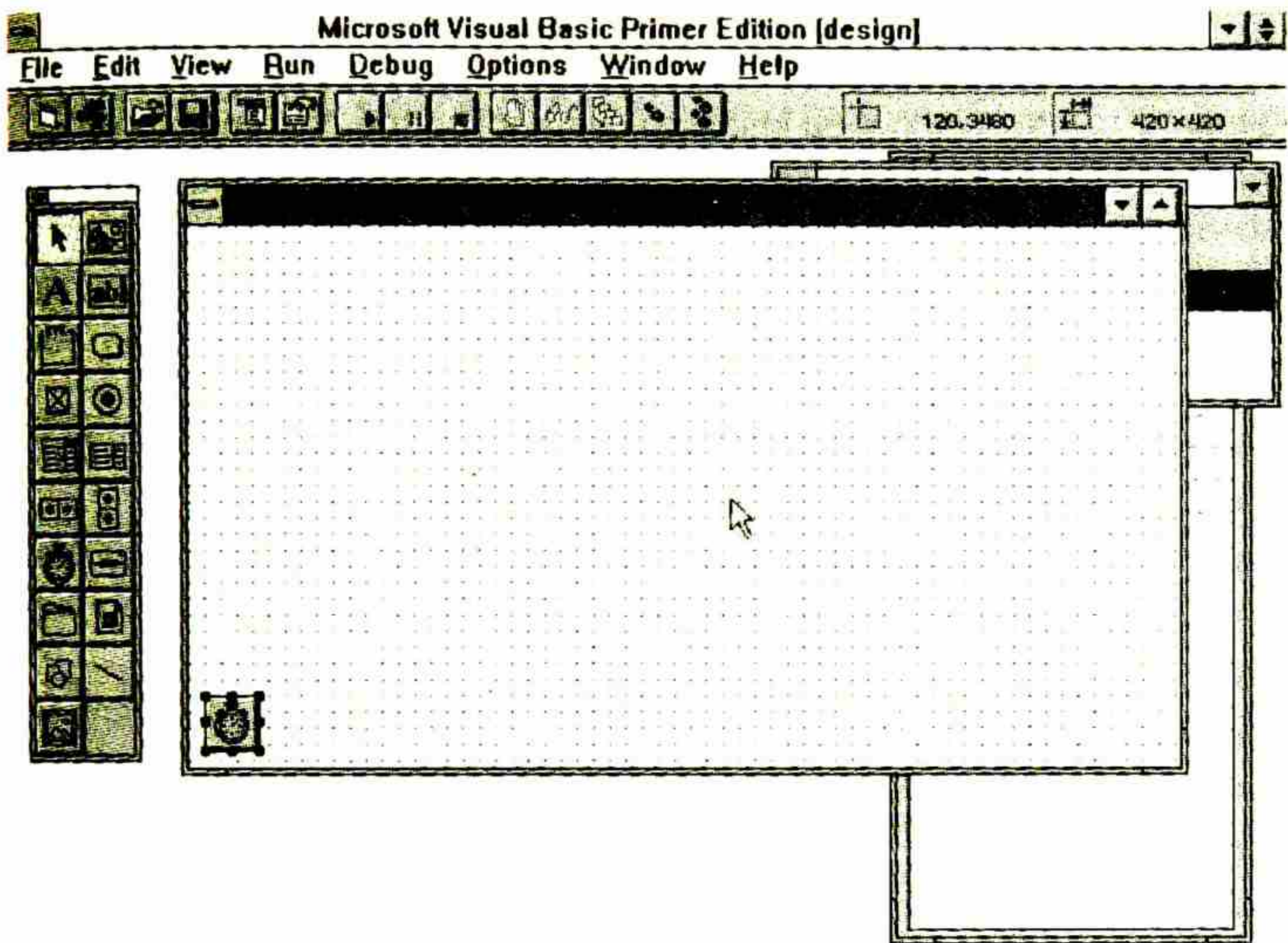
Hình 20.1. *Điều khiển bộ định thời là biểu tượng đồng hồ bấm giờ trên cửa sổ Toolbox.*

Vị trí đặt bộ định thời không quan trọng đã đành mà kích cỡ của nó cũng vậy. Không có vấn đề gì khi bạn điều chỉnh kích cỡ điều khiển bộ định thời, Visual Basic sẽ từ chối quá trình thay đổi kích cỡ đó. Không nên quan tâm đến kích cỡ của điều khiển bộ định thời bởi vì người dùng không nhìn thấy nó.

Hình 20.2 trình bày cửa sổ Form sau khi đặt điều khiển bộ định thời (Timer) vào góc trái dưới cửa sổ.

Lưu ý

Một trình ứng dụng bất kỳ đều có thể có nhiều điều khiển bộ định thời, như có nhiều điều khiển khác vậy.



Hình 20.2. *Mẫu biểu với một điều khiển bộ định thời .*

Bảng 20.1 trình bày danh sách giá trị thuộc tính của bộ định thời có tác dụng trong cửa sổ Property để thiết lập điều khiển. Các thuộc tính của điều khiển bộ định thời không nhiều. Bạn có thể nhớ rằng đa số thuộc tính của các điều khiển khác được thiết kế để trợ giúp quá trình chọn màu và định kích cỡ cho các điều khiển (như thuộc tính BackColor, Width, ...). Điều khiển bộ định thời không cần những thuộc tính này bởi vì người dùng không thấy điều khiển bộ định thời.

Khái niệm mới

1 mili giây (ms) bằng 1 phần ngàn giây.

Bảng 20.1. *Các thuộc tính điều khiển bộ định thời*

Thuộc tính	Mô tả
Enabled	Xác định điều khiển bộ định thời có thể đáp ứng biến cố.
Index	Xác định chỉ số mảng nếu lưu điểu khiển bộ định thời trong một mảng các điều khiển.
Interval	Xác định khoảng thời gian, tính theo mili giây, để một biến cố timer diễn ra. Giá trị Interval có thể biến thiên trong khoảng từ 1 đến 65535.
Left	Xác định khoảng cách tính bằng twip từ góc trái cửa sổ Form đến vị trí đặt điểu khiển bộ định thời.
Name	Tên của điều khiển. Trừ khi bạn thay đổi tên, Visual Basic sẽ gán các tên mặc định Timer1, Timer2, ... khi bạn thêm các điểu khiển bộ định thời.
Tag	Không được Visual Basic sử dụng. Thuộc tính này dành cho lập trình viên chú thích nút tùy chọn.
Top	Xác định khoảng cách tính bằng twip từ cạnh trên cùng cửa sổ Form tới vị trí điều khiển.

Thông thường, thuộc tính quan trọng nhất của bộ định thời là Interval bởi lẽ thuộc tính này dùng để xác định thời gian vận hành các thủ tục biến cố timer đã viết cho điều khiển bộ định thời.

Lưu ý

Hãy nhớ rằng số nguyên dương lớn nhất có thể lưu với kiểu dữ liệu nguyên (Integer) là 32767. Vì vậy, nếu muốn khởi tạo thuộc tính Interval của timer, hãy dùng biến kiểu số nguyên dài (Long Integer) hay kiểu dấu chấm động (Single hay Double) để giữ giá trị khởi tạo.

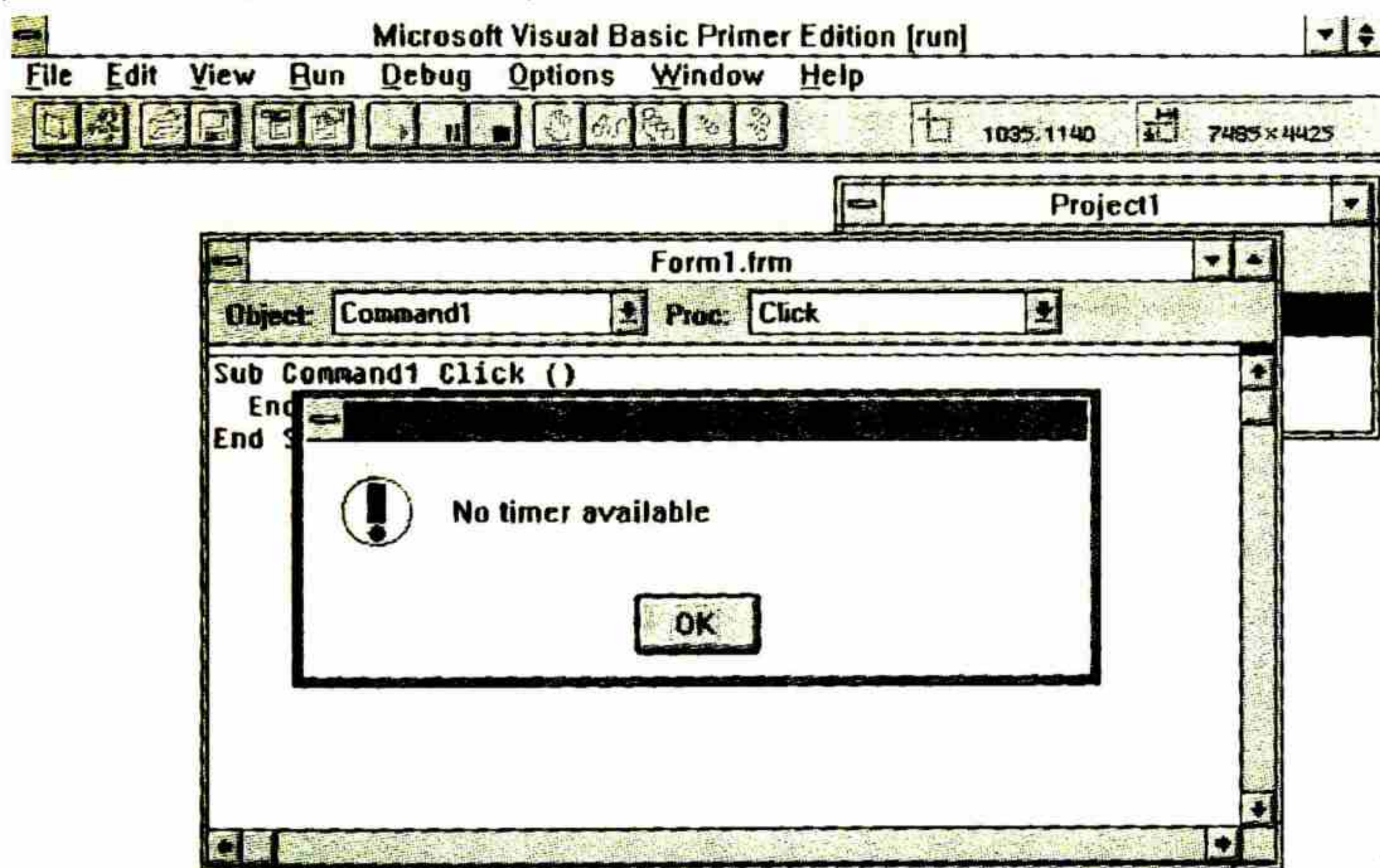
Trước khi học các lệnh sẽ đưa vào trong thủ tục biến cố, bạn cần hiểu rằng chỉ có một biến cố có tác dụng cho điều khiển bộ định thời. Điều khiển bộ định thời (Timer) có ít thuộc tính điều khiển và chỉ có thể viết một loại thủ tục biến cố.

Timer là biến cố dành cho điều khiển bộ định thời. Vì vậy, nếu đặt tên cho một điều khiển bộ định thời là tmrAlarm, biến cố cho điều khiển bộ định thời đó được gọi là tmrAlarm_Timer(). Để mở biến cố đó, bạn có thể nhấp đúp điều khiển bộ định thời, hay chọn mục Event từ danh sách xổ xuống Proc của cửa sổ Code để mở thủ tục biến cố Timer.

Lưu ý

Visual Basic phân biệt thủ tục biến cố Timer với hàm xây dựng sẵn Timer() dựa vào cách bạn sử dụng chúng.

Windows giới hạn số điều khiển bộ định thời (Timer) có thể thêm vào một trình ứng dụng tại một thời điểm. Nếu bạn thử thêm nhiều điều khiển bộ định thời hơn số Windows cho phép, Visual Basic sẽ hiển thị thông báo lỗi như trong Hình 20.3. Windows chỉ có thể hỗ trợ một số giới hạn các điều khiển bộ định thời (thông thường, Windows có thể xử lý 15 điều khiển bộ định thời trong một trình ứng dụng). Windows không thể đếm giá trị của tất cả thuộc tính Interval, nếu bạn sử dụng quá nhiều điều khiển bộ định thời, dễ dẫn đến xung đột và làm chậm tốc độ xử lý.



Hình 20.3. *Thông báo lỗi sẽ hiện ra nếu có quá nhiều điều khiển bộ định thời.*

Củng cố

Điều khiển bộ định thời (Timer) có ít thuộc tính và biến cố hơn các điều khiển khác. Không cần xác định thuộc tính định kích cỡ và màu đối với điều khiển bộ định thời bởi vì người dùng không thấy điều khiển bộ định thời và các thuộc tính này sẽ là vô nghĩa.

SỬ DỤNG ĐIỀU KHIỂN BỘ ĐỊNH THỜI

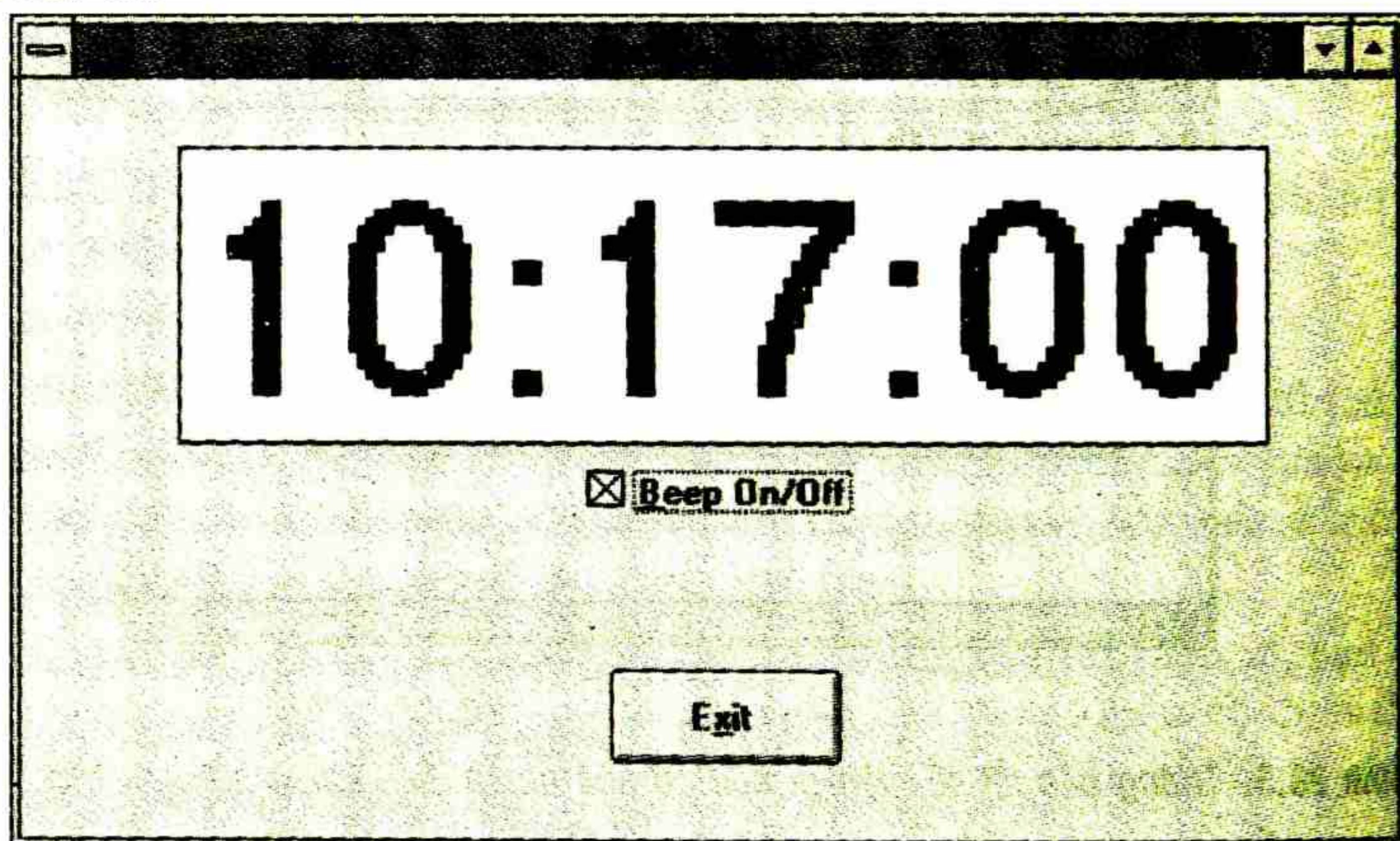
Khái niệm

Thuộc tính Interval xác định khoảng thời gian trôi qua giữa hai lần phát sinh biến cố bộ định thời. Khác với các điều khiển khác phát sinh biến cố để đáp ứng dữ liệu nhập vào của người dùng, bộ định thời lại tự phát sinh biến cố trong khoảng thời gian định trước.

Sử dụng điều khiển bộ định thời để thi hành một thủ tục biến cố tại thời điểm định trước. Khi đặt điều khiển bộ định thời trên một mẫu biểu, thuộc tính Interval của bộ định thời cho biết số mili giây sẽ trôi qua trước khi thủ tục biến cố Timer() của bộ định thời thi hành.

Giả sử bạn đặt một biến cố timer trên điều khiển và định thuộc tính Interval của bộ định thời bằng 10000. Nếu đặt tên cho biến cố timer là tmrSecond, thủ tục biến cố tmrSecond_Timer() sẽ thi hành cứ 10000 mili giây một lần, nói cách khác là 10 giây một lần.

Tóm tắt



Hình 20.4. *Xem thời gian cập nhật mỗi giây.*

Hình 20.4 trình bày ứng dụng CLOCK.VBP thực hiện cập nhật thời gian mỗi giây. Nút tùy chọn xác định bạn sẽ nghe tiếng bíp mỗi giây lúc đồng hồ cập nhật. Điều khiển bộ định thời có tên là tmrClock. Ví dụ 20.1 trình bày mã lệnh thủ tục biến cố tmrClock_Timer() sẽ cập nhật thời gian trên màn hình số và phát tiếng bíp.

Củng cố

Khởi tạo giá trị thuộc tính Interval để cho biết số mili giây sẽ trôi qua giữa hai lần xảy ra biến cố timer. Thủ tục biến cố Timer tự động thi hành sau từng khoảng thời gian trôi qua bằng với thời gian định trước trong thuộc tính Interval của bộ định thời.

Ví dụ 20.1. *Mã lệnh thủ tục biến cố Timer.*

```
1: Sub tmrClock_Timer ()
2: ' Beep every second if the check box
3: ' is set. Also, update the value of the clock
4: '
5: ' This event procedure executes every second
6: ' (or every 1,000 milliseconds) because the
7: ' timer control's Interval value is 1000
8: If (chkBeep.Value) Then
9: Beep
10: End If ' Don't beep if option button not set
11: ' Update the time inside the label
12: lblClock.Caption = Time$
13: End Sub
```

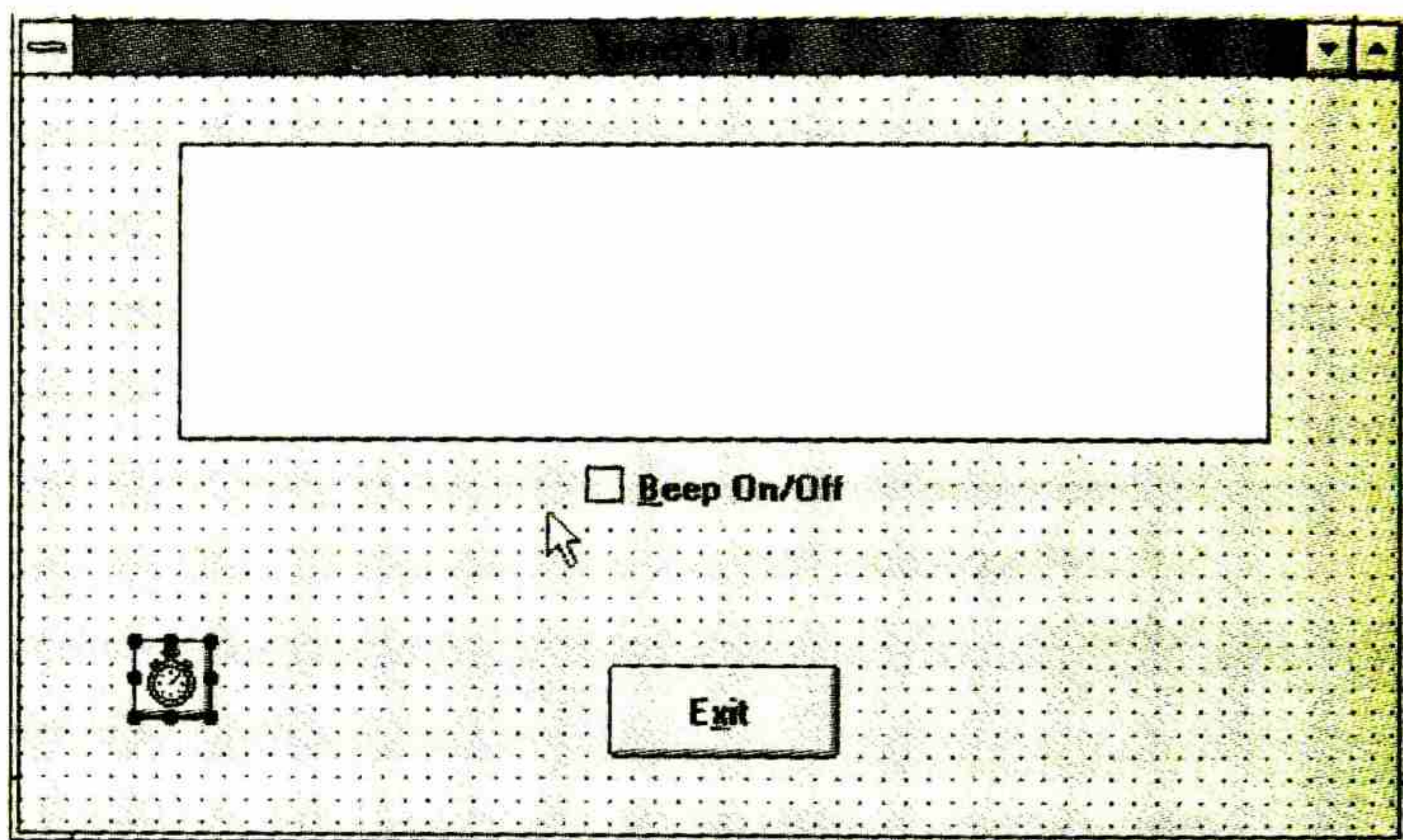
Phân tích

Dòng 1 cho biết tên điều khiển bộ định thời của trình ứng dụng là tmrClock. Do đó, thủ tục biến cố của nó là tmrClock_Timer(). Thuộc tính Interval của điều khiển bộ định thời thiết đặt bằng 1000 (mili giây), cho nên tmrClock_Timer() sẽ thi hành mỗi giây một lần.

Các dòng 8 đến 10 sẽ kiểm tra ô chọn để xem liệu người dùng muốn mỗi giây nghe tiếng bíp hay không. Nếu ô chọn được đánh dấu, khi đó thuộc tính Values của nó là True, dòng 9 sẽ thi hành.

Dòng 12 cập nhật giá trị của nhãn đồng hồ lớn ở giữa màn hình. Hàm Time$ truy xuất đồng hồ máy tính từng giây bởi vì thủ tục biến cố này sẽ thi hành mỗi giây.

Hình 20.5 trình bày cửa sổ Form của CLOCK.VBP trong thời gian thiết kế chương trình. Lưu ý điều khiển bộ định thời ở góc trái dưới màn hình. Điều khiển bộ định thời (Timer) này sẽ không xuất hiện khi người dùng chạy chương trình, nhưng tác dụng của điều khiển bộ định thời trong chương trình là điều chính yếu bởi vì điều khiển bộ định thời sẽ kích hoạt thủ tục biến cố duy nhất trong toàn bộ trình ứng dụng.



Hình 20.5. *Điều khiển bộ định thời sẽ không hiển thị khi chương trình vận hành.*

CÁC BIẾN CỐ ÍT XẢY RA

Khái niệm

Thuộc tính Interval giữ giá trị tối đa là 65535. 65535 xác định khoảng thời gian chỉ trên 1 phút. Nếu viết các biến cố timer sẽ thi hành trong khoảng thời gian ít nhất cũng hơn vài phút, bạn cần thêm mã lệnh duy trì quá trình vận hành biến cố mỗi lần.

Ví dụ, giả sử bạn muốn cảnh báo cho người dùng chọn lệnh File Save 5 phút một lần. Bạn cần một cái gì đó phải xảy ra mỗi lần trong một khoảng thời gian xác định, hãy dùng điều khiển bộ định thời. Để hiển thị hộp thông báo File Save 5 phút một lần, bạn thực hiện như sau:

1. Đặt điều khiển bộ định thời trên mẫu biểu của trình ứng dụng.

2. Xác định số mili giây có trong 5 phút. Có 1.000 mili giây trong một giây. Nghĩa là 1 phút có 60.000 mili giây. Vì vậy, có 300.000 mili giây trong 5 phút. Số mili giây tối đa là 65.535 (giá trị tối đa có thể nhận của thuộc tính Interval) cho nên bạn sẽ phải chia chu kỳ thời gian thành những khoảng thời gian nhỏ hơn có thể quản lý được theo mili giây, giây, phút.

Dễ dàng chia 5 phút thành 5 khoảng thời gian, mỗi khoảng 1 phút. Một phút bằng 60.000 mili giây. Vì vậy, bạn phải đặt thuộc tính Interval bằng 60000 và điều khiển việc hiển thị hộp thông báo File Save để hộp thông báo chỉ xuất hiện sau 5 lần thủ tục biến cố thi hành.

Tóm tắt

Ví dụ 20.2 trình bày thủ tục biến cố timer sẽ gọi lệnh message box 5 phút một lần. Biến cố timer thi hành mỗi phút, bởi vì giá trị Interval của bộ định thời là 60000. Cách đơn giản để hoàn thành quá trình hiển thị hộp thông báo 5 phút một lần là định nghĩa một biến module tên MinCount để đếm số phút trôi qua và gọi lệnh message box mỗi lần MinCount bằng 5. Lưu ý rằng ở đầu Ví dụ 20.2, bạn sẽ thấy thủ tục khai báo (General) định nghĩa một biến mức module tên MinCount. MinCount không mất giá trị cũ của nó giữa hai lần gọi thủ tục biến cố bởi vì MinCount là một biến module và không được định nghĩa cục bộ trong *tmrMinute_Timer()*.

Củng cố

Khi cần thi hành mã lệnh trong một khoảng thời gian lớn hơn 1 phút, hãy định nghĩa một biến module để giữ số giây hay phút đã trôi qua trong nhiều lần gọi thủ tục biến cố timer.

Ví dụ 20.2. *Thi hành một biến cố timer 5 phút một lần.*

```
' Inside the (general) procedure
' Define a module variable to keep
' track of one-minute time intervals
Option Explicit
Dim MinCount As Integer ' Begins at zero

Sub tmrMinute_Timer ()
```

```
8: ' Ensures that a message box appears only every
9: ' five minutes despite the fact that this
10: ' event procedure executes every passing minute
11: MinCount = MinCount + 1 ' Raise minute count
12: If (MinCount = 5) Then
13: MsgBox "It's time to save your file!"
14: MinCount = 0 ' Reset the minute count
15: End If
16: End Sub
```

Phân tích

Dòng 5 định nghĩa biến module tên MinCount. Tất cả các biến được định nghĩa đều có giá trị ban đầu là 0. Hãy nhớ rằng timer tên tmrMinute sẽ thi hành thủ tục con Timer trong ví dụ này có giá trị Interval bằng 60000, cho nên thủ tục biến cố tmrMinute_Timer() sẽ thi hành mỗi phút một lần.

Một khi số phút đếm bằng 5, nghĩa là 5 phút đã trôi qua kể từ khi biến MinCount được định nghĩa hay từ lần cuối cùng thủ tục biến cố này được thi hành. Dòng 13 sẽ hiển thị hộp thông báo và dòng 14 đặt lại biến đếm số phút bằng 0 để bắt đầu lại quá trình đếm mới. Tuy nhiên, khi số phút đếm chưa bằng 5, điều kiện If sẽ bằng False và thủ tục con sẽ không dừng, không hiển thị hộp thông báo.

BỘ ĐỊNH THỜI KHÔNG PHẢI LÀ ĐỒNG HỒ BÁO THỨC

Bạn không thể thiết đặt điều khiển bộ định thời để kích hoạt biến cố tại một thời điểm xác định. Điều khiển bộ định thời hoạt động không giống đồng hồ báo thức diễn ra tại một thời điểm xác định.

Khái niệm

Điều khiển bộ định thời sẽ thi hành thủ tục biến cố Timer của nó mỗi lần thời gian được định nghĩa trong Interval trôi qua. Tuy nhiên, điều khiển bộ định thời không hoạt động như đồng hồ báo thức, quá trình thi hành một thủ tục biến cố diễn ra lúc định lại thời gian. Tuy nhiên, bạn có thể lập trình thủ tục biến cố Timer để mô phỏng đồng hồ báo thức hay lịch để bàn.

Điều khiển bộ định thời (Timer) sẽ kích hoạt một biến cố lặp liên tục. Khi viết thủ tục biến cố timer, mã lệnh bên trong thủ tục phải có khả năng xử lý quá trình vận hành riêng của nó mỗi lần biến cố diễn ra. Nếu muốn thi hành một thủ tục con lúc định lại thời gian, bạn phải viết thủ tục biến cố để thi hành lặp (thực hiện mỗi lần khi đặt lại khoảng thời gian); sau đó mã lệnh phải kiểm tra xem liệu thời gian PC có bằng với thời gian định lại mà bạn đang chờ đợi.

Tóm tắt

Ví dụ 20.3 mô tả mã lệnh chương trình cần tạo đồng hồ báo thức như CLOCK.VBP. Khi chạy chương trình, bạn sẽ thấy nút lệnh tên Set Alarm. Lúc nhấp Set Alarm, một hộp nhận dữ liệu (Input Box) sẽ xuất hiện hỏi thời gian báo thức. Khi nhập vào thời gian đúng (chương trình sẽ lặp cho đến khi bạn nhập thời gian đúng hay chọn Cancel), thời gian báo thức sẽ xuất hiện bên phải nút lệnh như Hình 20.6 và đồng hồ báo thức sẽ vận hành khi đến thời gian qui định.

Củng cố

Khi viết chương trình đồng hồ báo thức hay lịch, hãy nhớ biến cố Timer luôn thi hành liên tục, ít nhất là 65.535 mili giây một lần bởi vì 65535 là giá trị tối đa có thể lưu trong thuộc tính Interval của điều khiển bộ định thời. Khi biến cố Timer thi hành, mã lệnh của bạn có thể kiểm tra thời gian hiện hành (hay ngày nếu viết chương trình lịch kiểm tra ngày) để xem liệu đã đến thời gian cần báo.

Ví dụ 20.3. *Mã lệnh định đồng hồ và đợi báo thức.*

```
' Define a module variable used in two
' different procedures. The module variable
' cannot lose its value between runs.
Option Explicit
Dim TimeSet As Variant

Sub cmdSet_Click ()
' Ask the user for an alarm time to set
' Keep looping until the user:
' Enters a valid time or
```

```
' clicks the Cancel command button
Do
TimeSet = InputBox("Enter a 24-hour time to wake up", "Time")
If (TimeSet = "") Then ' Check for Cancel
Exit Sub ' User pressed Cancel
End If
Loop Until IsDate(TimeSet) ' Keep asking if invalid
' If here, the user entered a value that can
' be converted to a valid time value. Now,
' convert the Variant data to a date/time type
TimeSet = CVDate(TimeSet)
' Display the wake-up time in a label
lblSet.Caption = "Alarm @ " & TimeSet
End Sub

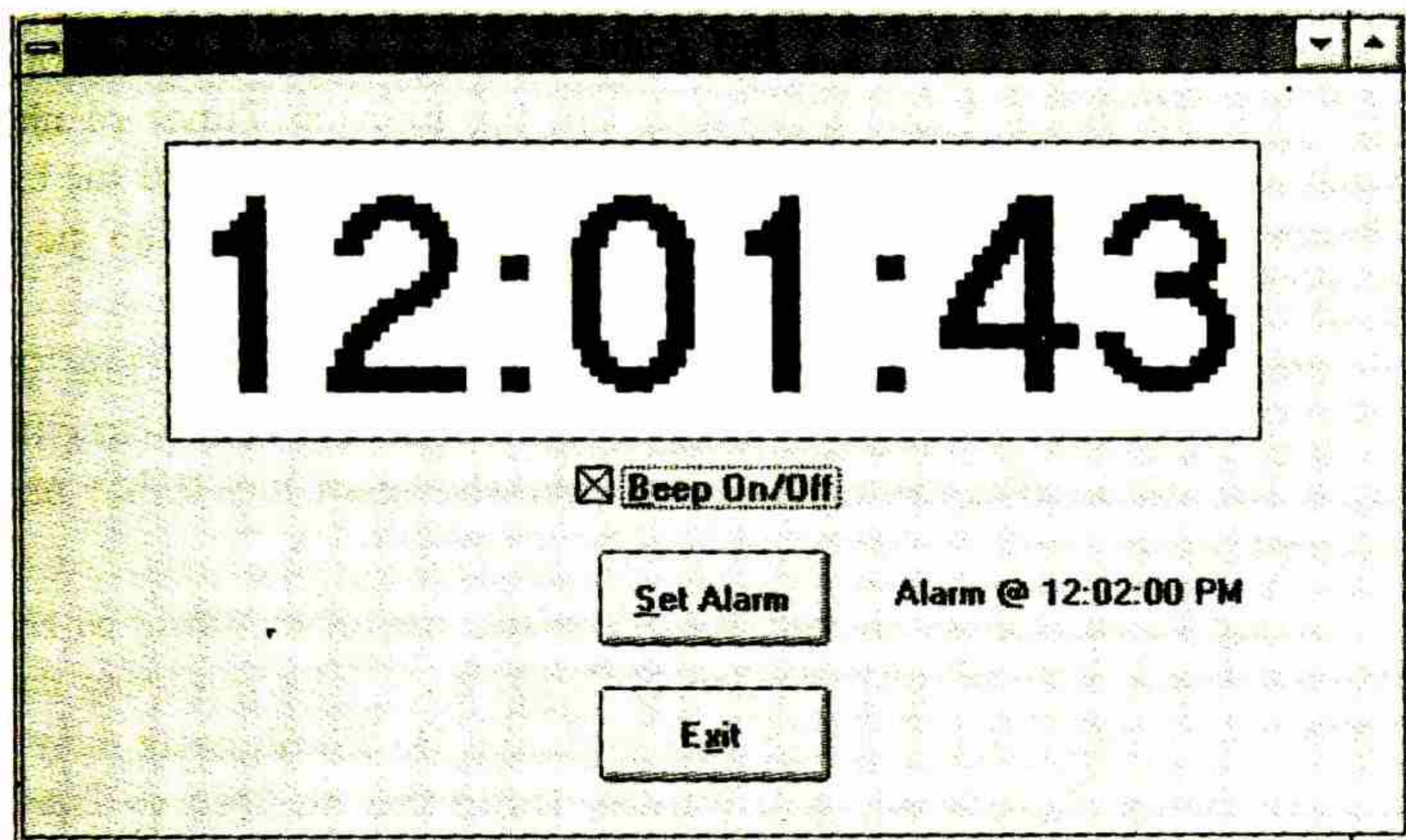
Sub tmrClock_Timer ()
' Sound an alarm if the set time has been hit
' just now or perhaps bypassed by a fraction
Dim ctr As Integer

If (chkBeep.Value) Then ' Second beep if needed
Beep
End If
lblClock.Caption = Time$ ' Update the screen's clock

' Sound the alarm if the wake up time has been
' passed (also, make sure a time has been set)
If (Time >= TimeSet) And (lblSet.Caption <> "No alarm") Then
For ctr = 1 To 50
Beep ' Wake up!!!
Next ctr
MsgBox "W-a-k-e U-p-!-!-!", MB_ICONEXCLAMATION, "Alarm"
lblSet.Caption = "No alarm" ' To keep alarm from resounding
End If
End Sub
```

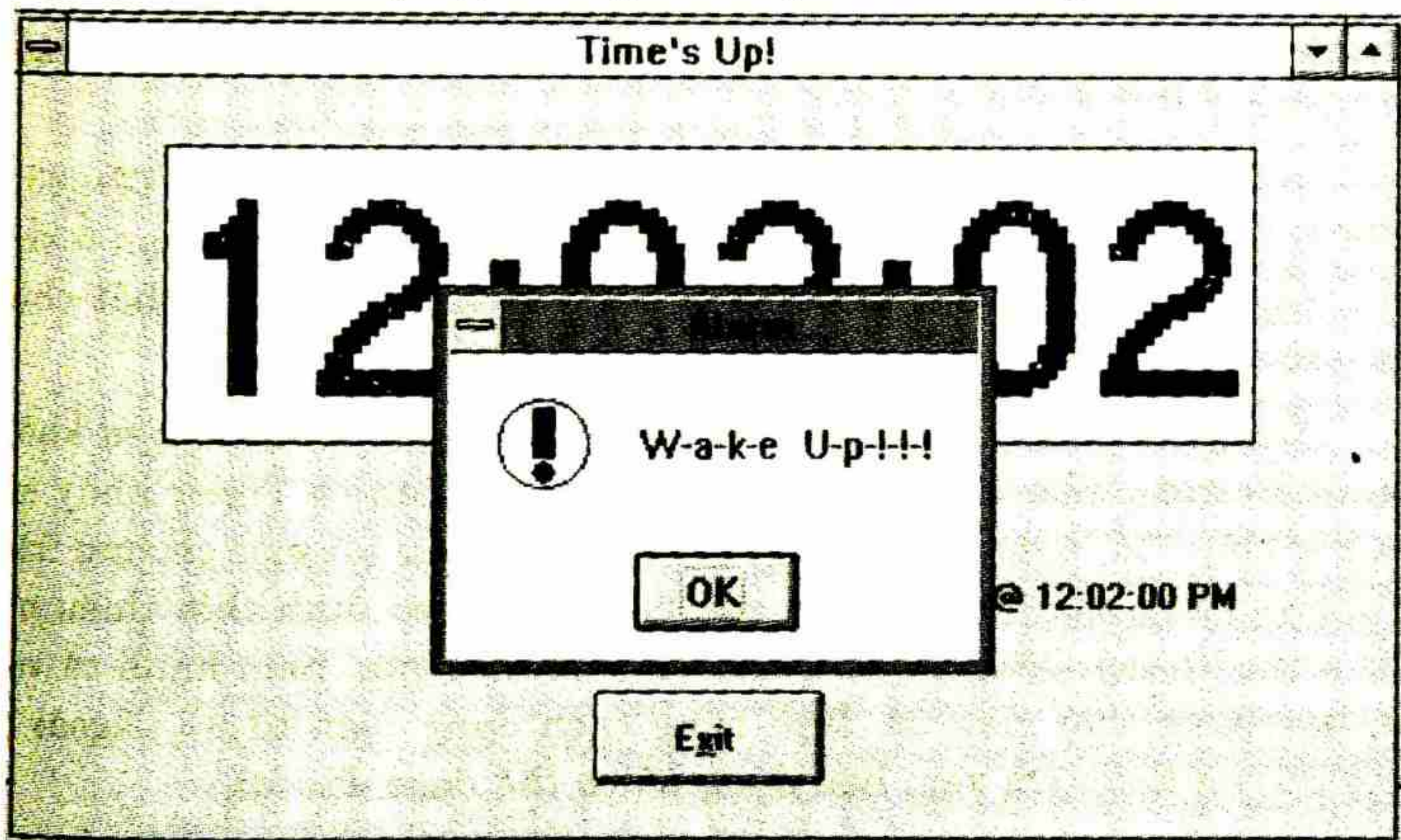
Kết quả

Hình 20.6 trình bày những gì diễn ra khi người dùng định thời gian để kích hoạt đồng hồ báo thức.



Hình 20.6. *Đồng hồ báo thức đã sẵn sàng hoạt động.*

Khi đồng hồ báo thức phát âm thanh, người dùng nghe chuỗi tiếng bíp và nhìn thấy hộp thông báo được mô tả trong Hình 20.7.



Hình 20.7. *Đúng giờ báo thức người dùng.*

Phân tích

Dòng 1 đến 5 là mã lệnh (General) định nghĩa một biến thủ tục tên TimeSet lưu thời gian báo thức người dùng. Lý do bạn phải biết định nghĩa biến module khi phân chia biến cố Timer, thay vì dùng một danh sách đối số để truyền giá trị, là vì Visual Basic phát sinh biến cố Timer yêu cầu Visual Basic phát sinh thủ tục biến cố Timer có nhận hay gửi các đối số. Vì vậy, biến bất kỳ mà bạn dùng trong thủ tục biến cố Timer phải là biến module hay toàn cục nếu muốn khởi tạo giá trị biến đó ở nơi khác và sử dụng biến đó trong thủ tục Timer.

Dòng 7 bắt đầu thủ tục hỏi người dùng thời gian báo thức. Thủ tục con bao gồm một vòng lặp từ dòng 12 đến 17 để hỏi người dùng thời gian và duy trì cho đến khi người dùng nhập thời gian hợp lệ hay nhấn nút lệnh Cancel.

Dòng 21 sau đó sẽ chuyển biến Variant của người dùng có chứa giá trị thời gian hợp lệ (hãy cảm ơn lệnh gọi hàm IsDate() trong dòng 17, đảm bảo người dùng đã nhập thời gian hợp lệ) thành dạng ngày để bạn có thể dùng so sánh với hàm Time$() trong thủ tục biến cố Timer() về sau. Dòng 23 sẽ thay đổi phụ đề mặc định của nút lệnh từ No Alarm thành Set Alarm cho quá trình thiết đặt thời gian báo thức mới.

Lưu ý

Nếu người dùng chỉ nhập vào một ngày không kèm theo thời gian, hàm IfDate() ở dòng 17 sẽ bỏ qua ngày. Không có cách kiểm tra thời gian bằng cách dùng một hàm xây dựng sẵn.

Dòng 26 bắt đầu thủ tục biến cố Timer. Các dòng từ 31 đến 33 sẽ kiểm tra liệu sẽ phát tiếng bíp lần thứ hai (người dùng đã chọn ô chọn trên mẫu biểu nếu muốn nghe tiếng bíp). Dòng 34 sẽ cập nhật chính xác quá trình hiển thị đồng hồ lớn trên màn hình từng giây.

Dòng 38 kiểm tra xem liệu đã đến giờ báo thức. Bởi vì sự không hoàn hảo trong Windows, có khả năng thủ tục biến cố Timer nhảy một lần hai giây. Có lẽ do người dùng tạm thời chuyển tới một chương trình Windows khác, mà đôi khi làm cho điều khiển bộ định thời thiếu mất một nhịp. Vì vậy, dòng 38 sẽ kiểm tra xem thời gian hiện hành có vượt quá hay bằng thời gian đã định bằng cách dùng toán tử >=. Ngoài ra, dòng 38 còn kiểm tra xem liệu đồng hồ có thể được đặt lại.

Nếu đúng giờ báo thức đồng hồ đã định, vòng lặp For từ dòng 39 đến 41 sẽ phát một dãy 50 tiếng bíp ra loa, và sau đó dòng 42 sẽ hiển thị một hộp thông báo đánh thức người dùng. Dòng 43 đặt lại phụ đề cho đồng hồ, cho nên chương trình không còn duy trì âm thanh báo thức cho những giây sau lệnh này.

Bài tập

Kiến thức tổng quát

1. Điều khiển nào đếm thời gian trôi qua?
2. Thuộc tính quan trọng nhất của điều khiển bộ định thời là gì?
3. Tại sao điều khiển bộ định thời không có các thuộc tính kích cỡ và màu sắc?
4. Người dùng sẽ thấy gì ở góc trái trên khi vận hành mẫu biểu chương trình có đặt một điều khiển bộ định thời ở vị trí đó?
5. Đúng hay Sai: Người dùng kích hoạt các biến cố timer.
6. Giá trị thời gian được tính trong thuộc tính Interval của điều khiển bộ định thời theo đơn vị nào?
7. Đúng hay Sai: Một trình ứng dụng có thể có nhiều nhất một điều khiển bộ định thời.
8. Mili giây là gì?
9. 10000 ms bằng bao nhiêu thời gian?
10. 60000 ms bằng bao nhiêu thời gian?
11. Tên tiền tố mà các lập trình viên thường gán cho điều khiển bộ định thời là gì?
12. Vấn đề gì có thể gặp nếu thử định thuộc tính Interval của bộ định thời cho một biến integer ?
13. Tên thủ tục biến cố duy nhất có hiệu lực trong điều khiển bộ định thời là gì ?
14. Đúng hay Sai: Bạn có thể định thuộc tính Interval của bộ định thời để kích hoạt một biến cố ở thời điểm xác định trước.
15. Tại sao đôi khi Visual Basic xảy ra lỗi nếu kiểm tra giá trị hàm Time$() khác kiểu thời gian chính xác?

Viết mã lệnh...

16. Hãy sửa đổi mã lệnh project 7 để hiển thị thời gian cập nhật mỗi giây. Quá trình cập nhật nên diễn ra mà không phụ thuộc múi giờ người dùng chọn.

Phần nâng cao

Viết một ứng dụng có 3 nút lệnh tên 10 Seconds, 30 Seconds, 45 Seconds. Khi người dùng nhấp một nút lệnh, hãy hiển thị một điều khiển đếm xuống dần số giây thích hợp. Khi bộ đếm hoàn thành, phát tiếng bíp 5 lần để báo người dùng quá trình đếm đã hoàn thành. Bạn có thể dùng ba điều khiển bộ định thời riêng biệt hoặc sửa đổi giá trị interval của bộ định thời phụ thuộc vào nút lệnh do người dùng chọn.

Bài thực hành 10

Tạo chương trình "Thế giới thực"

Tóm tắt

Chương này chỉ bạn cách thiết kế và bổ sung menu vào trình ứng dụng Visual Basic của bạn. Cửa sổ Menu Design cho phép bạn xác định giá trị thuộc tính cho các lệnh trên thanh menu và menu xổ xuống. Bằng cách dùng cửa sổ Menu Design, bạn có thể thêm các phím truy xuất nhanh và đường phân cách menu, xác định những đặc tính menu đặc biệt như một số mục menu được đánh dấu chọn. Quá trình phát sinh mã lệnh chương trình đáp ứng biến cố menu không khó hơn so với quá trình viết thủ tục biến cố đáp ứng thao tác nhấp nút lệnh.

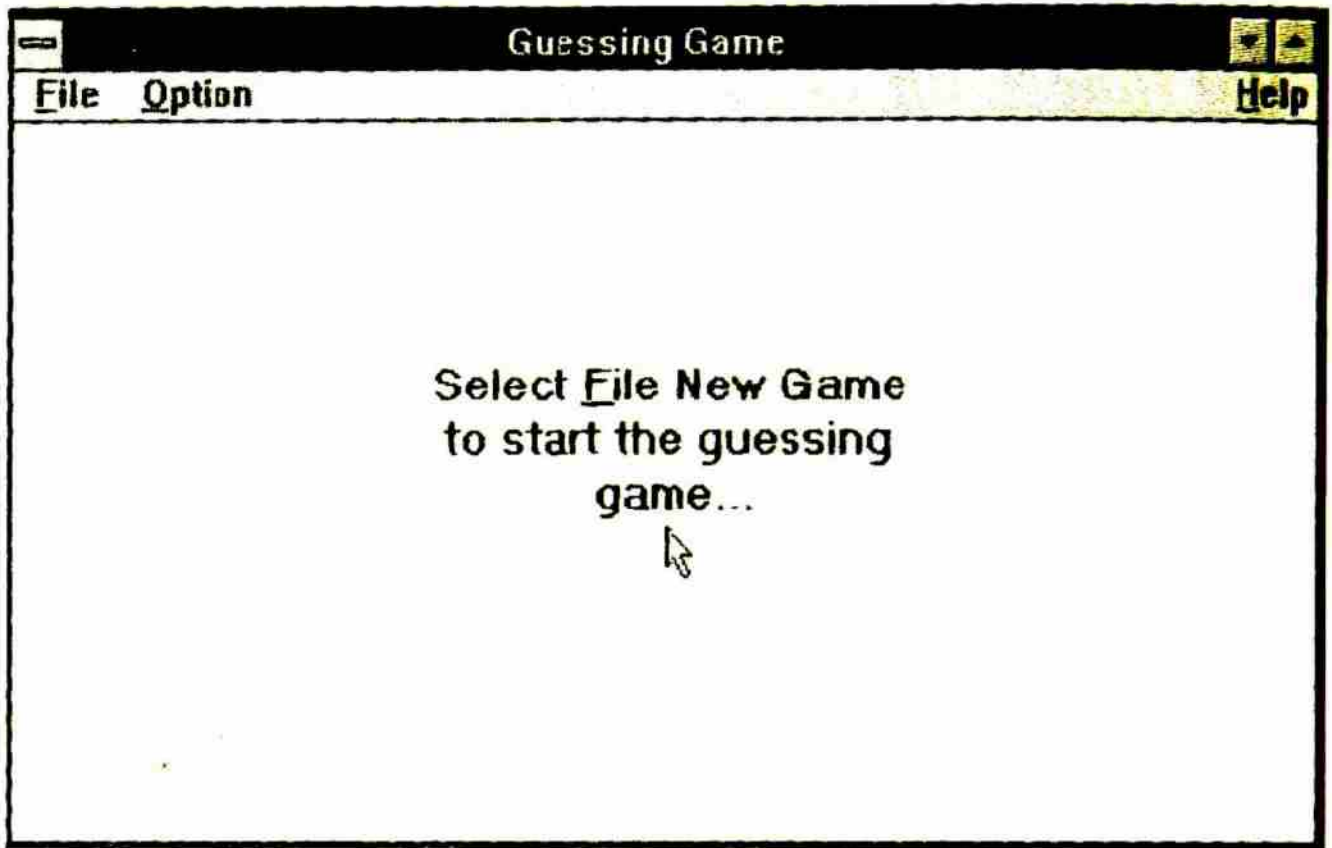
Để tạo menu, bạn cũng học cách làm việc với điều khiển bộ định thời. Như đã thấy ở bài trước, điều khiển bộ định thời (Timer) là điều khiển tự kích hoạt biến cố riêng của nó. Phụ thuộc vào giá trị thuộc tính Interval, điều khiển bộ định thời sẽ phát sinh biến cố trong khoảng thời gian tính theo mili giây. Mã lệnh thêm vào thủ tục biến cố Timer sẽ xác định cách xử lý biến cố. Bạn có thể đặt chuông báo và thi hành các hành động cập nhật điều khiển và màn hình xác định mỗi lần Timer phát sinh một biến cố nếu muốn.

Trong chương này, bạn đã tìm hiểu:

- Cách tạo menu cho trình ứng dụng Visual Basic.
- Lý do cửa sổ Menu Design thích hợp hơn nhiều so với quá trình xác định giá trị thuộc tính menu qua cửa sổ Property.
- Cách canh lề phải các lệnh menu nếu muốn thay đổi thanh menu một chút.
- Điều khiển bộ định thời sẽ phát sinh biến cố nào.
- Thời điểm đáp ứng biến cố Timer và khi nào sẽ bỏ qua biến cố đó.

Mô tả chương trình

Hình P10.1 minh họa cửa sổ Form đang mở PROJECT10.VBP. Bài thực hành này trình bày cách đoán số và ký tự có trong thanh menu với ba mục. Chương trình gồm ba thủ tục biến cố gắn với các lệnh menu.



Hình P10.1. *Khởi đầu bằng mẫu biểu đơn giản.*

Cửa sổ Menu Design không có gì hơn ngoài cửa sổ Property nâng cao dành cho menu. Biến cố menu để mã lệnh của bạn đáp ứng là các biến cố Click. Vì vậy, không có điều gì thực sự mới mẻ đối với mã lệnh được sử dụng trong chương trình. Chương trình sẽ phát sinh một số hay ký tự ngẫu nhiên khi người dùng chọn File New Game. Nếu tùy chọn Option Use Alphabetic Letters được chọn, chương trình sẽ phát sinh một ký tự ngẫu nhiên từ A đến Z. Nếu tùy chọn menu Option Use Alphabetic Letters không được chọn, chương trình sẽ phát sinh một số ngẫu nhiên từ 1 đến 100. Nhãn nhỏ ở cuối màn hình sẽ nhắc người dùng đoán một số hay ký tự khi quá trình đoán bắt đầu.

Tính ngẫu nhiên của chương trình

Bởi vì không có yêu cầu gì khác về mã lệnh đáp ứng biến cố menu, bài thực hành trong chương này sẽ giới thiệu một lệnh và hàm mới để phát sinh giá trị ngẫu nhiên và là chương trình đáng chú ý hơn. Bộ phát sinh số ngẫu nhiên chỉ là một phần của ngôn ngữ Visual Basic, nó không cho kết quả giống nhau mỗi khi thi hành cùng mã lệnh.

Hàm Rnd(), khi được dùng mà không có đối số, sẽ phát sinh một số ngẫu nhiên từ 0 đến 1. Tuy nhiên, nếu người dùng muốn đoán một số, chương trình cần một số ngẫu nhiên từ 1 đến 100, và nếu người

dùng muốn đoán một ký tự, chương trình cần một số ngẫu nhiên từ 65 đến 90 để làm đối số cho hàm Chr$() cho các ký tự từ A đến Z. Chương trình sẽ sử dụng công thức sau để phát sinh một số ngẫu nhiên từ giá trị thấp đến giá trị cao:

```
Int((LowValue + HighValue + 1) * Rnd + LowValue
```

Lệnh gán sau đây sẽ phát sinh một số ngẫu nhiên từ 65 đến 90, và lưu giá trị ký tự của số ASCII được phát sinh trong biến Letter:

```
Letter = Chr$(Int(26) * Rnd + 65) ' A through Z
```

Chỉ có một vấn đề với hàm Rnd. Hàm sẽ phát sinh cùng giá trị giống nhau mỗi khi thi hành chương trình sử dụng Rnd, trừ khi bạn thi hành câu lệnh sau đây trước tiên:

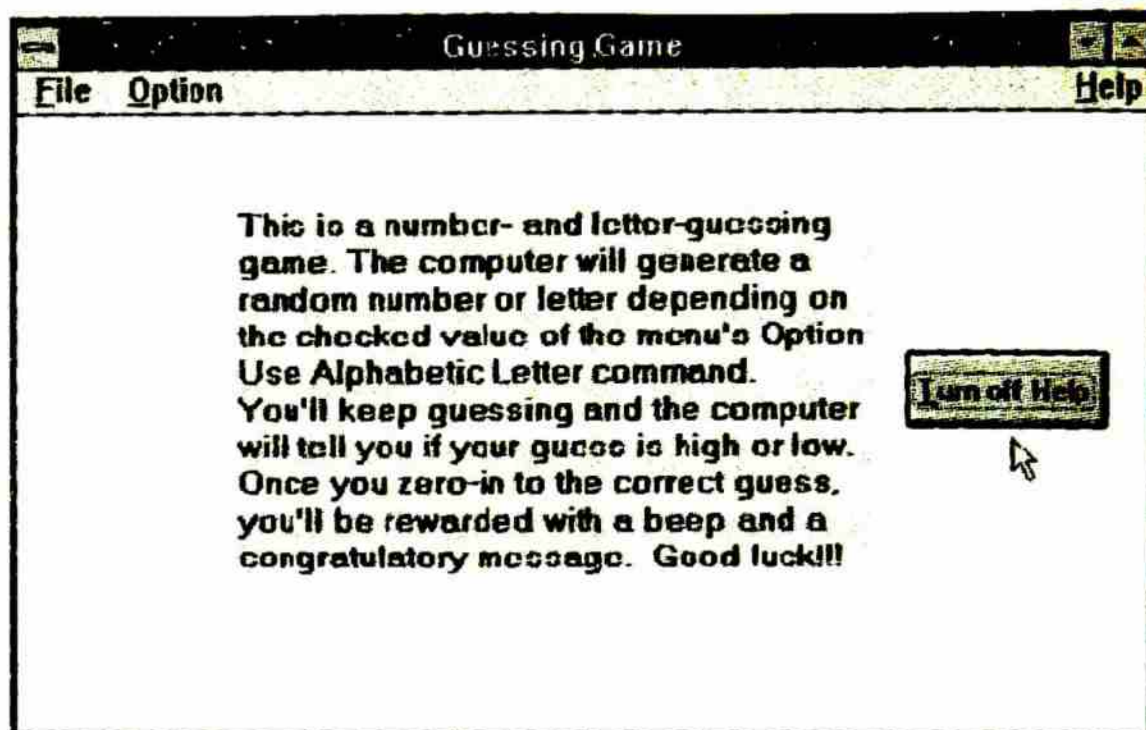
```
Randomize Timer
```

Lệnh Randomize yêu cầu một giá trị đơn. Giá trị đó hình thành bộ phát sinh số ngẫu nhiên bằng cách cung cấp giá trị cho số ngẫu nhiên được phát sinh đầu tiên. Nếu bạn đã sử dụng cùng giá trị phát sinh, hàm Rnd sẽ không thay đổi tập hợp giá trị sẽ phát sinh trong những lần chạy chương trình. Trò chơi đoán thường đưa ra tập hợp giá trị giống nhau để quá trình đoán luôn hấp dẫn. Bằng cách dùng Timer phát sinh bộ phát sinh số ngẫu nhiên bên trong, bạn có thể đảm bảo rằng Randomize hiếm khi làm việc với cùng giá trị hai lần, bởi vì giá trị phát sinh ngẫu nhiên là xác lập đồng hồ thời gian nội tại mà lệnh Randomize sẽ thi hành.

Menu Help

Bạn chú ý rằng menu Help được canh phải. Thủ tục biến cố Form_Load() nối ký tự backspace Chr$(8) với lệnh Help của thanh menu. Bài 19 có đề cập ký tự backspace sẽ canh phải các lệnh thanh menu.

Hình P10.2 minh họa thông báo trợ giúp và nút lệnh sẽ xuất hiện khi người dùng chọn lệnh Help. Thủ tục biến cố của lệnh Help minh họa trong Ví dụ P10.1, đặt thuộc tính Visible của nút lệnh và nhãn trợ giúp thành True để người dùng có thể thấy thông báo trợ giúp. Khi người dùng nhấp nút lệnh, thủ tục biến cố cmdHelp_Click() (được mô tả trong Ví dụ P10.1) sẽ thi hành để giấu nhãn trợ giúp và nút lệnh.



Hình P10.2. *Lệnh menu Help khái quát chương trình cho người dùng.*

Ví dụ P10.1. *Đáp ứng lệnh Help và tắt trợ giúp khi được yêu cầu.*

```
1: Sub mnuHelp_Click ()
2: ' Show the help message and the help command button
3: lblHelp.Visible = True
4: cmdHelp.Visible = True
5: End Sub
6:
7: Sub cmdHelp_Click ()
8: ' Hide the help message and the help command button
9: lblHelp.Visible = False
10: cmdHelp.Visible = False
12: End Sub
```

Mô tả

1: Mã lệnh thủ tục biến cố đáp ứng lệnh thanh menu Help phải là thủ tục Click.

2: Chú thích giải thích thủ tục.

3: Nhãn lớn sẽ hiển thị ở giữa màn hình.

(3: Help là nhãn bình thường không thể thấy được.)

4: Bật nút lệnh người dùng có thể sử dụng để giấu thông tin trợ giúp.

5: Kết thúc thủ tục.

6: Dòng trống phân tách các thủ tục.

7: Mã lệnh thủ tục biến cố đáp ứng nút lệnh tắt thông tin trợ giúp.

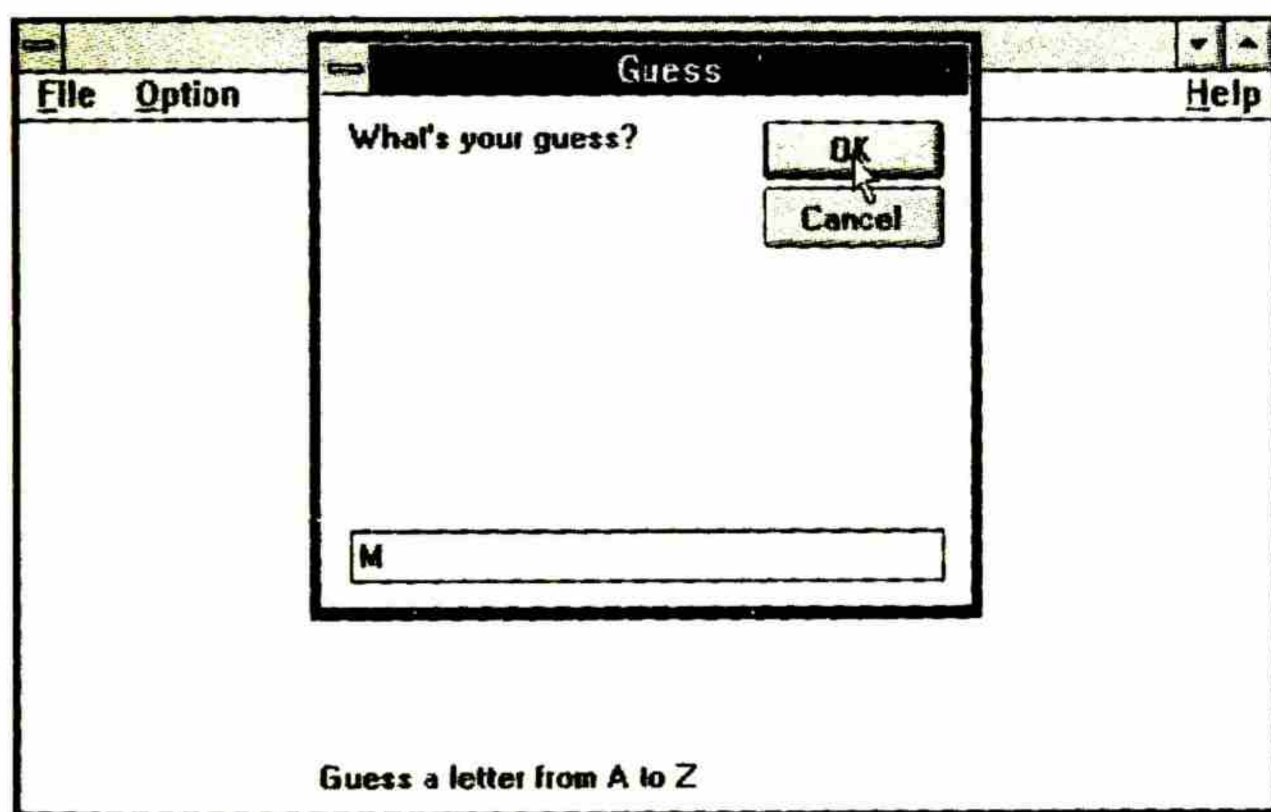
8: Chú thích giải thích thủ tục.

9: Tắt nhãn lớn đang hiển thị ở giữa màn hình.

10: Tắt nút lệnh người dùng có thể dùng để giấu thông tin trợ giúp.

11: Kết thúc thủ tục.

Nghiên cứu phần còn lại của mã lệnh



Hình P10.3. *Người dùng phải đoán ký tự từ A đến Z.*

Ví dụ P10.2 trình bày phần còn lại của chương trình, kể cả thủ tục biến cố Form_Load() canh phải lệnh Help và khởi động bộ phát sinh số ngẫu nhiên để những lệnh gọi hàm Rnd kế tiếp lúc thi hành chương trình sẽ trả về các giá trị khác nhau. Hình P10.3 mô tả chương trình đang chạy trong suốt quá trình đoán ký tự của người dùng. Chú ý rằng nhãn hiển thị ở cuối màn hình trong suốt quá trình chơi báo cho người dùng biết sẽ đoán ký tự hay số.

Ví dụ P10.2. *Mã lệnh cho ví dụ thực hành trò chơi đoán số hoặc ký tự.*

```
Sub Form_Load ()
' Begin with a number-guessing game
' Turn off the checkmark from the
' Options Use Alphabetic Characters menu
mnuOptionUseAlpha.Checked = False
' Right-justify the Help menu command
mnuHelp.Caption = Chr$(8) + mnuHelp.Caption
' Spin the internal randomizing wheel
Randomize Timer
End Sub

Sub mnuFileExit_Click ()
End
End Sub

Sub mnuFileNewGame_Click ()
' Call appropriate subroutine procedure depending
' on value of Option Use Alphabetic Characters

' Turn off opening label if it's still visible
If (lblOpening.Visible = True) Then
lblOpening.Visible = False
End If

If (mnuOptionUseAlpha.Checked) Then
Call GuessLetter
Else
Call GuessNumber
End If
' Turn back on opening label
lblOpening.Visible = True
```

```
End Sub
Sub mnuOptionUseAlpha_Click ()
' Reverse the state of the checkmark
mnuOptionUseAlpha.Checked = Not (mnuOptionUseAlpha.Checked)
End Sub
Sub GuessNumber ()
' User guesses a number
Dim Guess As Variant
Dim Number, GuessNum As Integer
' Update the instruction label
lblInstruct.Caption = "Guess a number from 1 to 100"
lblInstruct.Visible = True
' Find value from 1 to 100
Number = Int(100 * Rnd + 1)
Do
Guess = InputBox("What's your guess?", "Guess")
If (Guess = "") Then ' Check for Cancel
' Turn off instruction label
lblInstruct.Visible = False
Exit Sub ' User pressed Cancel
End If
GuessNum = Guess ' Convert to number
If (GuessNum > Number) Then
MsgBox "Your number is too high...", , "Wrong"
Else
If (GuessNum < Number) Then
MsgBox "Your number is too low...", , "Wrong"
End If
End If
```

```
Loop Until (GuessNum = Number)
Beep
MsgBox "You guessed " & Guess & "! Correct!!!", , "Lucky"
' Turn off instruction label
lblInstruct.Visible = False
End Sub

Sub GuessLetter ()
' User guesses a letter
Dim Letter, GuessLet As String

' Update the instruction label
lblInstruct.Caption = "Guess a letter from A to Z"
lblinstruct.Visible = True
' Find value from ASCII 65 – 90 ('A' to 'Z')
Letter = Chr$(Int(26) * Rnd + 65) ' A through Z
Do
GuessLet = InputBox("What's your guess?", "Guess")
If (GuessLet = "") Then ' Check for Cancel
' Turn off instruction label
lblInstruct.Visible = False
Exit Sub ' User pressed Cancel
End If
' Convert user's letter to uppercase
GuessLet = UCase(GuessLet)
If (GuessLet > Letter) Then
MsgBox "Your letter is too high...", , "Wrong"
Else
If (GuessLet < Letter) Then
MsgBox "Your letter is too low...", , "Wrong"
End If
End If
Loop Until (GuessLet = Letter)
```

```
98:
99: ' Here is user guessed number
100: Beep
101: MsgBox "You guessed " & GuessLet & "! Correct!!!", , "Lucky"
102:
103: ' Turn off instruction label
104: lblInstruct.Visible = False
105: End Sub
```

Mô tả

1: Mã lệnh thủ tục biến cố đang mở mẫu biểu sẽ thi hành trước khi người dùng thấy mẫu biểu.

2: Chú thích giải thích thủ tục.

3: Chú thích tiếp theo.

4: Chú thích tiếp theo.

5: Ấn định quá trình chơi là đoán ký tự.

6: Chú thích giải thích mã lệnh.

7: Canh phải tùy chọn menu Help.

8: Chú thích giải thích mã lệnh.

9: Dùng đồng hồ nội tại để khởi động bộ phát sinh số ngẫu nhiên.

(9: Hãy chắc chắn khởi động bộ phát sinh số ngẫu nhiên trước khi hiển thị những giá trị ngẫu nhiên.)

10: Kết thúc thủ tục.

11: Dòng trống phân tách thủ tục.

12: Mã lệnh cho lệnh menu File Exit.

13: Kết thúc quá trình thi hành chương trình.

14: Kết thúc thủ tục.

15: Dòng trống phân tách thủ tục.

16: Mã lệnh của lệnh menu File New Game.

17: Chú thích giải thích thủ tục.

18: Chú thích tiếp theo.

19: Dòng trống giúp phân tách các phần chương trình.

20: Chú thích giải thích thủ tục.

21: Tắt việc hiển thị nhãn Press File New Game nếu nó đang bật.

22: Giấu nhãn.

23: Kết thúc lệnh If.

24: Dòng trống giúp phân tách từng phần chương trình

25: Xem liệu người dùng muốn đoán ký tự.

26: Thi hành thủ tục Subroutine cho phép người dùng đoán ký tự.

27: Nếu người dùng muốn đoán số...

28: Thi hành thủ tục Subroutine cho phép người dùng đoán số.

29: Kết thúc lệnh If.

30: Chú thích giải thích mã lệnh.

31: Bây giờ người dùng đã hoàn thành trò chơi, bật nhãn Select File New Game.

32: Kết thúc thủ tục.

33: Dòng trống phân tách từng phần chương trình.

34: Mã lệnh của lệnh menu Option Use Alphabetic Letters.

35: Chú thích giúp giải thích thủ tục.

(35: Chọn hay không chọn lệnh menu khi nhấp.)

36: Bật hoặc tắt dấu chọn của lệnh menu.

37: Kết thúc thủ tục.

38: Dòng trống giúp phân tách thủ tục.

39: Thủ tục Subroutine được gọi khi người dùng muốn đoán số.

40: Chú thích giải thích thủ tục.

41: Định nghĩa biến lưu giá trị trả về của hàm InputBox().

42: Định nghĩa biến lưu số người dùng đoán và số được phát sinh ngẫu nhiên trong trò chơi đoán.

43: Dòng trống giúp phân tách từng phần thủ tục.

44: Chú thích giải thích mã lệnh.

45: Thay đổi nhãn sẽ xuất hiện ở cuối màn hình trong suốt quá trình đoán của người dùng.

46: Hiển thị nhãn đang đoán.

47: Chú thích giải thích mã lệnh.

48: Phát sinh một số ngẫu nhiên từ 1 đến 100.

(48: Trừ khi bạn chuyển giá trị rơi vào một vùng khác, Visual Basic sẽ phát sinh một số trong khoảng từ 0 đến 1.)

49: Dòng trống giúp phân tách từng phần mã lệnh.

50: Bắt đầu vòng lặp đoán.

51: Lấy kết quả đoán của người dùng.

52: Kiểm tra xem liệu người dùng đã nhấn nút lệnh Cancel khi hiển thị hộp nhận dữ liệu.

53: Chú thích giải thích mã lệnh.

54: Tắt nhãn đoán.

55: Kết thúc sớm thủ tục con đoán số.

56: Kết thúc lệnh If.

57: Đổi giá trị trả lời của người dùng từ Variant thành số nguyên.

58: Kiểm tra xem liệu giá trị đoán của người dùng là quá lớn.

59: Nếu người dùng đoán số quá lớn, hãy báo cho người dùng biết.

60: Ngược lại người dùng đoán số không lớn.

61: Kiểm tra liệu người dùng có đoán số quá nhỏ.

62: Nếu người dùng đoán số quá nhỏ, hãy báo cho người dùng biết.

63: Kết thúc lệnh If.

64: Kết thúc lệnh If.

65: Duy trì quá trình hỏi người dùng giá trị sẽ đoán cho đến khi người dùng đoán đúng câu trả lời.

66: Phát tiếng bíp cho người dùng biết họ đã đoán đúng.

67: Hiện thông báo cho người dùng biết đã đoán đúng.

68: Chú thích giúp giải thích mã lệnh.

69: Giấu nhãn chỉ thị đang đoán.

70: Kết thúc thủ tục.

71: Dòng trống giúp phân tách các thủ tục.

72: Thủ tục Subroutine được gọi khi người dùng muốn đoán ký tự.

73: Chú thích giải thích thủ tục.

74: Định nghĩa biến lưu ký tự người dùng đoán và ký tự ngẫu nhiên.

75: Dòng trống giúp phân tách các phần thủ tục.

76: Chú thích giải thích mã lệnh.

77: Thay đổi nhãn sẽ xuất hiện ở cuối màn hình trong suốt quá trình đoán của người dùng.

78: Hiển thị nhãn đang đoán.

79: Chú thích giải thích mã lệnh.

80: Phát sinh ký tự ngẫu nhiên thuộc các ký tự từ A đến Z.

81: Bắt đầu vòng lặp đoán.

82: Lấy ký tự đoán của người dùng.

83: Kiểm tra xem liệu người dùng đã nhấn nút lệnh Cancel lúc hiển thị hộp nhận dữ liệu.

84: Chú thích giải thích mã lệnh.

85: Tắt nhãn đang đoán.

86: Kết thúc sớm thủ tục con đang đoán.

87: Kết thúc lệnh If.

88: Chú thích giải thích mã lệnh.

89: Đổi ký tự người dùng đoán thành chữ hoa.

(89: Hàm UCase$() duy trì giá trị ký tự người đoán ở dạng chữ hoa.)

90: Kiểm tra xem liệu người dùng đoán ký tự cao hơn phạm vi qui định.

91: Nếu người dùng đoán ký tự vượt quá phạm vi qui định, hãy báo cho người dùng biết.

92: Ngược lại người dùng đoán ký tự không cao quá phạm vi qui định.

93: Kiểm tra xem liệu người dùng đoán ký tự dưới phạm vi qui định.

94: Nếu người dùng đoán ký tự dưới phạm vi qui định, hãy báo cho người dùng biết.

95: Kết thúc lệnh If.

96: Kết thúc lệnh If.

97: Duy trì quá trình hỏi ký tự đoán từ người dùng cho đến khi người dùng đoán đúng câu trả lời.

98: Dòng trống giúp phân tách các phần mã lệnh.

99: Chú thích giúp giải thích mã lệnh.

100: Phát tiếng bíp báo cho người dùng biết đã đoán đúng.

101: Thông báo cho người dùng biết đã đoán đúng.

102: Dòng trống giúp phân tách các phần mã lệnh.

103: Chú thích giúp giải thích mã lệnh.

104: Giấu nhãn chỉ thị đoán.

105: Kết thúc thủ tục.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Chương kế tiếp sẽ chỉ bạn cách gửi văn bản ra máy in và mang lại cho bạn sự thích thú bằng cách vẽ những hình ảnh trên mẫu biểu người dùng.

Chương XI

Bài 21

Sử dụng máy in

- Lưu ý
- In trong Windows
- Báo cho người dùng biết máy in đã sẵn sàng
- Giới thiệu đối tượng Printer
- Phương thức Print
- Quá trình khởi tạo in
- Dấu phân trang

Bài này trình bày cách sử dụng máy in trong trình ứng dụng Visual Basic. Visual Basic sẽ liên lạc với trình điều khiển máy in trong Windows để bạn có thể gửi văn bản và hình ảnh đến máy in.

Khác với nhiều công việc khác trong Visual Basic, việc xuất kết quả ra máy in có thể là một quá trình buồn tẻ. Đáng ngạc nhiên ở chỗ điểm yếu cũng chính là điểm mạnh của Visual Basic: quá trình in đòi hỏi bạn phải gửi danh sách lệnh tương đối dài đến máy in để mô tả chính xác cách xuất dữ liệu ra máy in. Visual Basic cho phép bạn điều khiển và quản lý quá trình in dễ dàng như thực hiện với các điều khiển. Đôi khi bạn cần điều khiển trực tiếp cách in, kể cả phông chữ của từng ký tự mà trình ứng dụng gửi tới máy in. Vì vậy, quá trình điều khiển máy in dài dòng cho phép bạn điều khiển tất cả chi tiết in chính xác tuyệt đối.

LƯU Ý

Khái niệm

Visual Basic 6.0 hỗ trợ lệnh Print trên menu File. Tuy nhiên, bạn vẫn có thể in báo cáo từ trình ứng dụng.

Lệnh Print trên menu File luôn cho bạn in mã lệnh trong một trình ứng dụng. Thêm vào đó, bạn có thể in mẫu biểu đồ họa trong chương trình cũng dễ dàng như in văn bản mô tả về mẫu biểu. Các tài liệu như thế có thể giúp bạn tìm hiểu và nắm bắt quan điểm thiết kế khi muốn sửa đổi trình ứng dụng về sau.

Củng cố

Có thể in mẫu biểu và tài liệu mã lệnh từ lệnh Print trên menu File trong Visual Basic hoặc bằng cách áp dụng các thuộc tính và phương thức sẽ trình bày trong bài này.

IN TRONG WINDOWS

Khái niệm

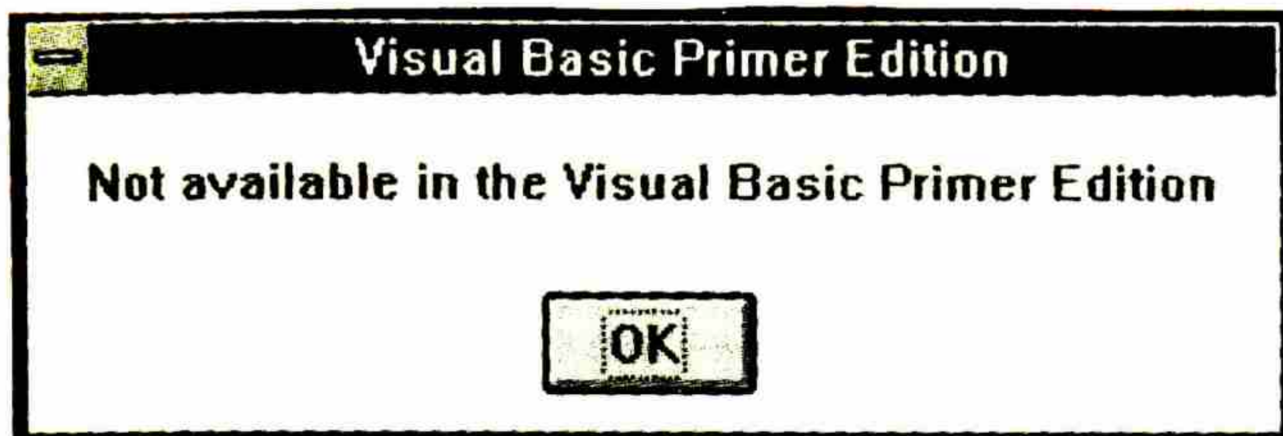
Khi trình ứng dụng xuất dữ liệu ra máy in, Windows sẽ chặn các lệnh này. Ít khi có việc gửi trực tiếp dữ liệu ra máy in, Visual Basic sẽ gửi dữ liệu in thực sự tới trình quản lý in trong Windows.

Khái niệm mới

Trình quản lý in trong Windows điều khiển tất cả dữ liệu được in trong Windows.

Trình quản lý in trong Windows xác định cách thức trình bày dữ liệu in trong tất cả chương trình Windows. Vì vậy, khi trình ứng dụng Visual Basic tìm cách gửi dữ liệu in trực tiếp tới máy in, trình quản lý in trong Windows sẽ chặn các lệnh này, và có thể thay đổi dữ liệu xuất ra trước khi gửi cho máy in.

Trình quản lý in trong Windows biết cách liên lạc với bất kỳ máy in nào được Windows hỗ trợ. Windows có thể nhận diện hàng trăm loại máy in khác nhau, và hầu hết các máy in này đòi hỏi những lệnh thích ứng riêng. Mỗi chương trình bạn mua phải hỗ trợ loại máy in mà bạn đang có và yêu cầu nhiều không gian đĩa hơn không gian làm việc của nó. Chương trình thường đắt tiền hơn bởi vì các nhà phát triển phần mềm phải mất thời gian lập trình để xuất dữ liệu kết quả ra từng loại máy in mà chương trình có thể sử dụng.



Hình 21.1. *Trình quản lý in trong Windows tập hợp tất cả dữ liệu xuất của chương trình và quản lý các máy in riêng biệt.*

Hiếm khi có yêu cầu một phần mềm phải hỗ trợ tất cả các loại máy in, trình quản lý in trong Windows đòi hỏi mỗi phần mềm chỉ cần hỗ trợ một loại máy in để in dữ liệu và phải thuộc loại máy in mà trình quản lý in trong Windows hỗ trợ. Nếu trình ứng dụng bạn viết cần in ra kết quả, Visual Basic sẽ xuất dữ liệu ra một mẫu biểu mà trình quản lý in trong Windows yêu cầu. Hình 21.1 sẽ trình bày trình ứng dụng Visual Basic xuất dữ liệu trực tiếp ra trình quản lý in trong Windows. Sau đó trình quản lý in trong Windows sẽ chuyển dữ liệu xuất thành các lệnh riêng.

Giả sử bạn gắn vào máy tính của bạn cả máy in lazer lẫn máy in kim. Không có trình quản lý in trong Windows, bạn cần phải cung cấp hai bộ lệnh máy in cho trình ứng dụng Visual Basic mà bạn viết. Với trình quản lý in trong Windows, bạn chỉ cần cung cấp một bộ lệnh in chung. Trước khi chạy trình ứng dụng, bạn có thể sử dụng các lệnh có hiệu lực trong trình quản lý in trong Windows để chọn một trong hai máy in của bạn. Khi chạy chương trình, Windows sẽ chuyển kết quả Visual Basic thành những lệnh cần cho bất kỳ máy in nào được chọn.

Củng cố

Trình quản lý in trong Windows đơn giản hóa quá trình liên lạc với tất cả các loại máy in khác nhau. Trình ứng dụng Visual Basic chỉ cần gửi dữ liệu ra trình quản lý in trong Windows mà không cần biết loại máy in nào sẽ thực hiện in dữ liệu đó. Trình quản lý in trong Windows biết cách liên lạc với tất cả máy in được Windows hỗ trợ, và chuyển dữ liệu xuất của trình ứng dụng Visual Basic ra dạng dữ liệu được yêu cầu ứng với máy in đã chọn.

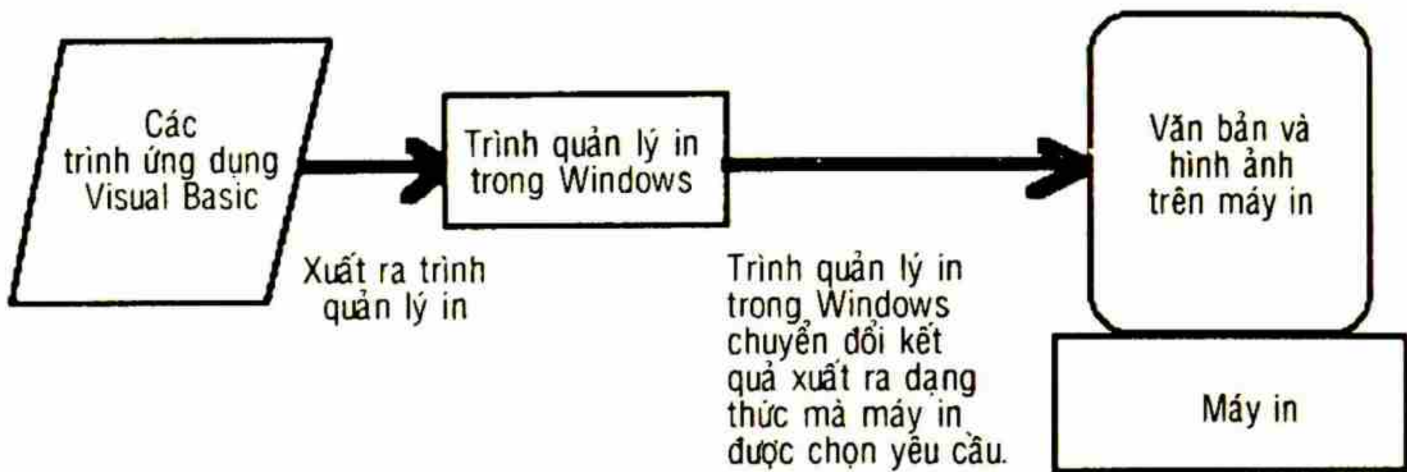
BÁO CHO NGƯỜI DÙNG BIẾT MÁY IN ĐÃ SẴN SÀNG

Khái niệm

Người dùng có thể không biết khi trình ứng dụng của bạn bắt đầu in thì máy in phải sẵn sàng.

Khái niệm mới

Online ***nghĩa là máy in đã sẵn sàng in.***



Hình 21.2. *Qui trình in dữ liệu ra máy in.*

Luôn nhớ rằng người dùng bật máy in, chắc chắn máy in có giấy và bảo đảm máy in đang online. Nếu máy in của người dùng chưa được bật và có sẵn giấy, người dùng sẽ nhận thông báo lỗi từ trình quản lý in trong Windows tương tự như mô tả trong Hình 21.3.

Tóm tắt

Thủ tục Function trong Ví dụ 21.1 cung cấp cho bạn lệnh MsgBox() rất có ích để bạn đưa vào chương trình trước khi in.

Củng cố

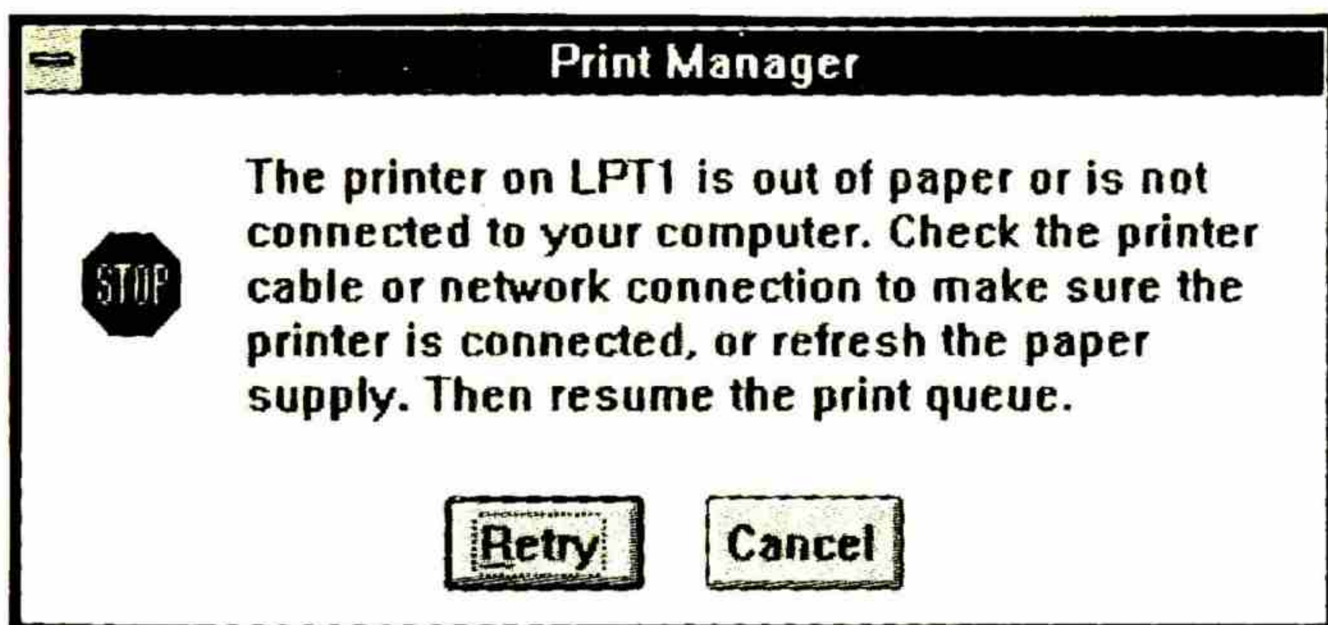
Quá trình cảnh báo người dùng khi in để đảm bảo người dùng đặt giấy và bật máy in ở chế độ online.

Ví dụ 21.1. *Cảnh báo người dùng trước khi in.*

```
1: Function PrReady()
2: ' Make sure the user is ready to print
3: Dim IsReady As Integer
4: IsReady = MsgBox("Make sure the printer is ready", 1, "Printer Check")
5: If (IsReady = 2) Then
6: PrReady = 0 ' A Cancel press returns a False value
7: End If
8: PrReady = 1 ' User pressed OK so return True
9: End Function
```

Kết quả

Hình 21.3 mô tả hộp thông báo sẽ xuất hiện trong Ví dụ 21.1.



Hình 21.3. *Người dùng bây giờ biết cần chuẩn bị máy in để in.*

Phân tích

Sau khi đọc thông báo và trả lời hộp thông báo trong dòng 4, giá trị trả về của thủ tục sẽ xác định liệu người dùng muốn xem kết quả (giả sử người dùng đã chuẩn bị xong máy in cho quá trình in) hay hủy bỏ quá trình in. Giá trị trả về bằng 0 (nghĩa là False) hay 1 (nghĩa là True) có thể được chọn từ thủ tục khác dựa trên cách đáp ứng của người dùng:

```
If PrReady() Then ' If function is true
 Call PrintRoutine
End If
```

GIỚI THIỆU ĐỐI TƯỢNG Printer

Khái niệm

Các trình ứng dụng Visual Basic gửi kết quả in đến một đối tượng Visual Basic đặc biệt gọi là đối tượng Printer. Đối tượng Printer hỗ trợ nhiều giá trị thuộc tính và phương thức mà bạn sẽ xác định để xem kết quả in.

Từ khóa Printer xác định đối tượng máy in mà trình ứng dụng của bạn sẽ gửi kết quả ra. Không có điều khiển Printer trên cửa sổ Toolbox. Tất cả truy xuất Printer phải diễn ra thông qua mã lệnh Visual Basic.

Ghi chú

Các lệnh ứng dụng của bạn gửi đến đối tượng Printer là lệnh chung cho máy in trong Windows. Trình quản lý in trong Windows sẽ chuyển những lệnh chung này thành các lệnh tương ứng với một máy in xác định. Vì vậy, bạn chịu trách nhiệm về những gì bạn muốn in và để trình quản lý in trong Windows chịu trách nhiệm về cách xuất kết quả ra máy in.

Khi tìm hiểu một đối tượng mới, chẳng hạn điều khiển nút lệnh (Command Button), bạn đã làm quen về các thuộc tính liên quan đến đối tượng đó. Trước khi sử dụng đối tượng Printer, bạn nên xem xét các thuộc tính có hiệu lực với đối tượng Printer để biết loại công việc nào bạn sẽ thực hiện với kết quả in trong Visual Basic. Tất cả thuộc tính của đối tượng Printer được liệt kê trong Bảng 21.1.

Khái niệm mới

Pixel ***là điểm có thể định địa chỉ nhỏ nhất trên màn hình hay máy in.***

Bảng 21.1. *Các thuộc tính của đối tượng Printer.*

Thuộc tính	Diễn giải
CurrentX	Lưu cột in theo chiều ngang tính từ góc trái trên trang, được đo theo đơn vị twip hay theo tỉ lệ được định nghĩa bởi thuộc tính Scale.
CurrentY	Lưu hàng in theo chiều dọc tính từ góc trái trên trang, được đo theo đơn vị twip hay theo tỉ lệ được định nghĩa bởi thuộc tính Scale.
DrawMode	Xác định dạng hình bạn vẽ trên máy in.
DrawStyle	Xác định kiểu đường do trình ứng dụng của bạn vẽ.
DrawWidth	Xác định độ rộng đường được vẽ, từ 1 (mặc định) đến 32767 pixel.
FillColor	Xác định màu hình dạng được in. Xác định độ đậm nhạt của hình được in không màu.
FillStyle	Chứa mẫu hình được in.
FontBold	Có giá trị True hoặc False để xác định liệu kết quả được in kế tiếp sẽ ở dạng đậm.
FontCount	Xác định số phông chữ được khởi tạo của máy in hiện hành.
FontItalic	Lưu giá trị True hoặc False để xác định liệu kết quả xuất ra kế tiếp ở dạng in nghiêng.
FontName	Lưu tên phông chữ hiện hành đang được dùng với dữ liệu xuất ra.
Fonts	Gồm bảng giá trị được lưu trong mảng điều khiển. Fonts(0) đến Fonts(FontCount – 1) lưu các tên phông chữ được khởi tạo trên máy tính.
FontSize	Kích cỡ phông chữ hiện hành tính theo point.
FontStrikeThru	Có giá trị True hoặc False để xác định liệu kết quả xuất kế tiếp sẽ được in với một đường gạch ngang.
FontTransparent	Có giá trị True hoặc False để xác định liệu kết quả xuất kế tiếp sẽ trong suốt.
FontUnderline	Có giá trị True hoặc False để xác định liệu kết quả xuất kế tiếp sẽ được gạch dưới.

ForeColor	Xác định màu nền của văn bản và hình ảnh được in (giấy sẽ xác định màu nền).
hDC	Một thiết bị Windows xử lý các lệnh gọi thủ tục Windows cao cấp.
Height	Chiều dài trang in hiện hành tính theo twip.
Page	Cho biết số trang hiện hành đang được in và được Visual Basic cập nhật tự động.
ScaleHeight	Xác định chiều cao theo đơn vị ScaleMode mà mỗi hình sẽ in ra.
ScaleLeft	Xác định khoảng cách tính theo đơn vị qui định trong ScaleMode kể từ lề trái trang đến nơi dữ liệu sẽ in ra.
ScaleMode	Đơn vị đo cho tất cả dữ liệu kết quả sẽ in.
ScaleTop	Xác định khoảng cách đầu trang đến nơi dữ liệu được in ra tính theo đơn vị đo qui định trong thuộc tính Scale Mode.
ScaleWidth	Xác định độ rộng hình theo đơn vị qui định trong ScaleMode sẽ được in ra.
TwipsPerPixelX	Xác định độ cao màn hình tính theo twip mà mỗi điểm của máy in (được gọi là pixel) chiếm.
TwipsPerPixelY	Xác định độ rộng màn hình tính theo twip mà mỗi điểm máy in hay pixel chiếm.
Width	Kích cỡ độ rộng trang (tính theo đơn vị twip).

Có nhiều thuộc tính Printer, như đã trình bày trong Bảng 21.1. May mắn thay, bạn chỉ dùng một vài thuộc tính cho phần lớn nhu cầu in của bạn. Các thuộc tính Printer liên quan về phông chữ sẽ gặp rất nhiều trong tài liệu in và đó là điều tất nhiên.

Ghi chú

Các thuộc tính và phương thức Printer liên quan đến hình ảnh sẽ không được đề cập trong bài này. Một khi bạn làm chủ các hình ảnh trong bài tiếp theo, bạn sẽ hiểu nhiều hơn về các thuộc tính Printer liên quan đến hình ảnh. Đa số thuộc tính của đối tượng Printer dùng để xuất hình ảnh cao cấp. Trong trình ứng dụng kinh điển, ít khi bạn thay đổi các thuộc tính này bởi vì những giá trị mặc định làm việc rất ăn ý với yêu cầu báo cáo thông thường.

Không giống đa số đối tượng điều khiển khác trong Visual Basic, phương thức của đối tượng Printer quan trọng hơn nhiều các giá trị thuộc tính của nó. Bảng 21.2 trình bày danh sách đầy đủ các phương thức được hỗ trợ bởi đối tượng Printer trong Visual Basic.

Bảng 21.2. *Các phương thức của đối tượng Printer*

Phương thức	Diễn giải
Circle	Vẽ đường tròn, hình oval hay đường cong.
EndDoc	Chuyển tài liệu hiện hành đến trình quản lý in để in ra.
Line	Vẽ các đường và hộp trên trang.
NewPage	Gửi một dấu phân trang tới kết quả in để kết quả tiếp đó sẽ in ở trang kế tiếp.
Print	In dữ liệu số và văn bản ra máy in.
PSet	Vẽ một điểm trên kết quả được in.
Scale	Xác định tỉ lệ sử dụng kết quả in.
TextHeight	Xác định chiều cao của văn bản theo tỉ lệ định trong thuộc tính Scale.
TextWidth	Xác định độ rộng của văn bản theo tỉ lệ định trong thuộc tính Scale.

Mách nước

Các phương thức phổ biến nhất của đối tượng Printer là Print, EndDoc, và NewPage. Một khi nắm vững ba phương thức này, bạn ít khi sử dụng bất kỳ một phương thức nào khác.

Củng cố

Có nhiều thuộc tính và phương thức có hiệu lực cho đối tượng Printer. Trừ khi bạn cần gửi hình ảnh đến máy in bằng cách dùng những khả năng đồ họa cao cấp, bạn chỉ cần hai hay ba thuộc tính và phương thức là có thể hoàn tất những yêu cầu in của bạn.

PHƯƠNG THỨC Print

Khái niệm

Phương thức Print của đối tượng Printer xử lý hầu hết các dữ liệu được in ra. Print hỗ trợ nhiều dạng khác nhau. Với phương thức Print, bạn có thể in thông báo, biến, hằng và biểu thức ra máy in.

Phương thức Print là phương thức được dùng thường xuyên nhất trong Visual Basic. Nắm rõ phương thức Print, tức là bạn đã làm chủ phương thức in quan trọng nhất.

Ghi chú

Hãy nhớ một phương thức không có gì cả ngoài lệnh áp dụng cho một đối tượng riêng biệt. Ví dụ, phương thức AddItem là phương thức bạn đã dùng trong sách để thêm các mục vào điều khiển hộp danh sách (List Box) và hộp combo (Combo Box).

Sau đây là dạng thức của phương thức Print:

[Printer.]Print [Spc(*n*) | Tab(*n*)] *Expression* [; | ,]

Dạng thức của phương thức Print làm nó trở nên khó hiểu hơn thực tế, nhưng phần phương thức Print xuất hiện ở bên phải từ khóa Print sẽ đưa ra một số giải thích. Các phần kế tiếp sẽ giải thích những tùy chọn khác nhau có hiệu lực trong phương thức Print.

In các hằng số

Phương thức Print in hằng chuỗi và số rất dễ dàng. Để in một hằng chuỗi hay số, hãy đặt hằng chuỗi hay số đó vào bên phải phương thức Print. Những phương thức sau sẽ gửi các số 1, 2, và 3 vào dữ liệu được in ra:

```
printer.Print 1
Printer.Print 2
Printer.Print 3
```

Khi quá trình thi hành gặp ba dòng mã lệnh này, Visual Basic sẽ gửi số 1, 2, và 3 đến đối tượng Printer với mỗi số xuất hiện trên một dòng. Mỗi phương thức Print sẽ gửi một ký tự trở lại đầu dòng và ký tự xuống dòng vào đối tượng Printer. Phương thức Print không kèm theo gì cả trên một dòng như sau:

```
printer.Print
```

Ghi chú

Visual Basic luôn thêm khoảng trống trước tất cả các số dương được in trên trang. Khoảng trống là nơi dấu + tưởng tượng sẽ xuất hiện.

Phương thức Print sau sẽ gửi 2 dòng văn bản tới đối tượng Printer:

```
printer.Print "Visual Basic makes writing programs"
Printer.Print "for Windows easy."
```

Khi trình quản lý in trong Windows nhận hai dòng dữ liệu xuất ra, kết quả sau sẽ xuất hiện trên trang giấy in:

```
Visual Basic makes writing programs
for Windows easy.
```

In biến và điều khiển

Ngoài các hằng, phương thức Print có thể in nội dung biến và điều khiển. Các lệnh sau sẽ khởi tạo một biến chuỗi và một biến nguyên, sau đó in nội dung các biến ra máy in:

```
FirstName = "Charley"
Age = 24
Printer.Print FirstName
Printer.Print Age
```

Sau đây là kết quả in ra với các phương thức Print này:

```
Charley
 24
```

Ghi chú

Hãy nhớ rằng Visual Basic sẽ không gửi gì cả đến đối tượng Printer cho đến khi mã lệnh có chứa phương thức Print thi hành. Bạn sẽ chèn phương thức Print ở vị trí thích hợp trong các thủ tục mã lệnh nơi yêu cầu in kết quả. Ví dụ, với nút lệnh Print Report, thủ tục Click của nút lệnh đó sẽ chứa các phương thức Print.

In biểu thức

Nếu bạn chỉ có thể in chuỗi, hằng số và biến, phương thức Print sẽ bị giới hạn. Tuy nhiên thực tế, phương thức Print không bị giới hạn.

Bạn có thể kết hợp các hằng, các biến và các biểu thức vào bên phải phương thức Print để đưa ra kết quả in phức tạp hơn nhiều. Phương thức Print sau sẽ in số 31:

```
printer.Print 25 + (3 * 2)
```

Biểu thức có thể bao gồm cả biến, điều khiển lẫn các hằng số, chẳng hạn như:

```
printer.Print Factor * lblWeight.Caption + 10
```

Nếu bạn muốn gửi các ký tự đặc biệt ra máy in, có thể thực hiện điều đó bằng hàm Chr$(). Biểu thức sau sẽ đưa ra một thông báo bao gồm cả dấu nháy kép trong các chuỗi được in:

```
printer.Print "She said, " & Chr$(34) & "I do." & Chr$(34)
```

Khi quá trình thi hành gặp dạng phương thức Print này, trình quản lý in sẽ gửi ra máy in kết quả sau :

```
She said, "I do."
```

Ghi chú

Bạn không thể in các dấu nháy kép mà không dùng hàm Chr$(). Thông thường, Visual Basic cấm các dấu nháy kép khi in các hằng chuỗi.

In nhiều giá trị

Khái niệm mới

Vùng in (Print zone) ***gồm 14 cột trên trang.***

Khi cần in nhiều giá trị trên một dòng, bạn có thể thực hiện điều đó bằng cách phân biệt các giá trị này với dấu chấm phẩy (;) và dấu phẩy (,). Dấu chấm phẩy bắt các giá trị kế tiếp xuất hiện ngay bên phải giá trị vừa xuất ra. Dấu phẩy bắt giá trị kế tiếp xuất hiện trong vùng in kế tiếp.

Hai thông báo sau sẽ được in trên các dòng khác nhau:

```
printer.Print "The sales were "
Printer.Print 4345.67
```

Bằng cách dùng dấu chấm phẩy, bạn có thể ép các giá trị này in cạnh nhau:

```
printer.Print "The sales were "; 4345.67
```

Dấu chấm phẩy cũng hoạt động để cho biết ký tự xuống dòng và trở về đầu dòng.

Phương thức Print sau kết thúc với một dấu chấm phẩy ở cuối:

```
printer.Print "The company name is ";
```

Dấu ; ở cuối cho biết đầu in của máy in ở cuối thông báo cho dữ liệu xuất kế tiếp. Vì vậy, lệnh Print kế tiếp sẽ in dữ liệu xuất ngay bên phải thông báo, cạnh kết quả in trước đó:

```
printer.Print lblComName.Caption ' Complete the line
```

Dấu ; tạo thuận lợi cho quá trình in nhiều giá trị với các kiểu dữ liệu khác nhau trên cùng dòng. Phương thức Print sai sẽ in tất cả dữ liệu của nó trên cùng dòng kết quả:

```
printer.Print "Sales:"; totSales; "Region:"; RegNum
```

Dấu phẩy đôi khi vẫn dùng để bắt các giá trị kế tiếp in trong vùng in kế tiếp. Phương thức Print sau sẽ in các tên, mỗi tên chiếm 14 khoảng trống trên cùng dòng:

```
printer.Print DivName1, DivName2, DivName3, DivName4
```

Không phân biệt tên dài hay ngắn, tên kế tiếp sẽ in trong vùng in kế tiếp. Phương thức in trước sẽ cho kết quả tương tự như sau:

```
North NorthWest South SouthWest
```

Mách nước

Khi in danh sách các số hay chuỗi ngắn, dấu phẩy cho phép bạn dễ dàng canh thẳng từng cột.

Sử dụng phông chữ

Phần lớn máy in tương thích trong Windows hỗ trợ nhiều loại phông chữ khác nhau. Các thuộc tính liên quan đến phông chữ thường có ích cho quá trình in các tiêu đề và thông báo dữ liệu xuất đặc biệt khác với kích cỡ và kiểu phông chữ đặc biệt.

Bạn có thể thêm những hiệu ứng đặc biệt vào văn bản được in của bạn bằng cách thiết đặt các thuộc tính điều chỉnh phông chữ từ Bảng 21.1. Ví dụ, đầu tiên mã lệnh sau sẽ đặt phông chữ đối tượng Printer có kích cỡ 72 point, nghiêng và đậm (kích cỡ in tương ứng với 1 inch), và sau đó in một thông báo:

```
printer.FontBold = True
Printer.FontItalic = True
Printer.FontSize = 72
Printer.Print "I'm learning Visual Basic!"
```

Ghi chú

Các thuộc tính phông chữ ảnh hưởng đến dữ liệu xuất ra kế đó. Vì vậy, nếu bạn in nhiều dòng văn bản và sau đó thay đổi kích cỡ phông chữ, văn bản mà bạn đã in sẽ không bị ảnh hưởng. Visual Basic chỉ in dữ liệu xuất ra kế tiếp với phông chữ mới.

Chèn khoảng trống với Spc() và Tab()

Phương thức Print hỗ trợ các hàm Spc() và Tab() được nhúng trong phương thức, giúp chương trình của bạn điều khiển dữ liệu xuất ra. Hàm Spc() trả lại số khoảng trống trong dữ liệu xuất được xác định bởi đối số trong hàm Spc(). Phương thức Print sau sẽ in 10 khoảng trống giữa họ và tên:

```
printer.Print FirstName; Spc(10); LastName
```

Đối số mà bạn gửi tới hàm Tab() nhúng trong phương thức Print sẽ xác định vị trí cột ký tự được in kế tiếp sẽ xuất hiện. Trong phương thức Print sau, ngày sẽ xuất hiện ở cột 50 trên trang:

```
printer.Print Spc(50); DateGenerated
```

Ví dụ này cho thấy nếu bạn in giá trị trước hàm Spc() và Tab(), bạn sẽ phân cách giữa các hàm và các giá trị được in bằng dấu chấm phẩy.

Mách nước

Spc() và Tab() cho bạn điều khiển quá trình thêm khoảng trống dễ dàng hơn cách dấu phẩy và dấu chấm phẩy.

Tóm tắt

Ví dụ 21.2 trình bày mã lệnh tính và in giá hai căn nhà và tiền thuế.

Củng cố

Dùng Spc() và Tab() điều khiển khoảng trống dữ liệu xuất ra máy.

Ví dụ 21.2. *Dùng hàm Spc() và Tab().*

```
1: Tax1 = TaxRate * HouseVal1
2: Tax2 = TaxRate * HouseVal2
3:
4: TotalVal = HouseVal1 + HouseVal2
5: TotTaxes = TaxRate * TotalVal
6:
7: Printer.Print "House Value"; Tab(20); "Tax"
8: Printer.Print Format$(HouseVal1, "Currency");
9: Printer.Print Tab(20); Format$(Tax1, "Currency")
10: Printer.Print Format$(HouseVal2, "Currency");
11: Printer.Print Tab(20); Format$(Tax2, "Currency")
12:
13: Printer.Print ' Prints a blank line
14:
15: Printer.Print "Total tax:"; Spc(5); Format$(TotTaxes, "Currency")
```

Kết quả

Sau đây là kết quả bạn có thể nhìn thấy trên giấy sau khi Ví dụ 21.2 thi hành:

```
House Value Tax
$76,578.23 $9,189.39
$102,123.67 $12,254.81
Total tax: $21,444.20
```

Phân tích

Hàm Tab(20) trong các dòng 7, 9, và 11 đảm bảo cột thứ hai chứa thông tin thuế sẽ được canh thẳng. Lưu ý rằng dấu chấm phẩy ở cuối dòng 8 và 10 cho phép bạn tiếp tục phương thức Print trên các dòng kế tiếp mà không ép những giá trị phương thức Print dài trên cùng một dòng.

Dòng 13 sẽ in một dòng trống. Dòng 15 sử dụng hàm Spc() để chèn 5 khoảng trống giữa tiêu đề và tổng số thuế.

QUÁ TRÌNH KHỞI TẠO IN

Khái niệm

Quá trình in vật lý không bắt đầu cho đến khi tất cả dữ liệu xuất ra được chuyển tới trình quản lý in, hay cho đến khi trình ứng dụng của bạn thực hiện phương thức EndDoc.

Khi bạn gửi phương thức Print đến trình quản lý in qua đối tượng Printer, trình quản lý in sẽ thiết kế trang hay nhiều trang dữ liệu xuất ra nhưng sẽ không gửi dữ liệu xuất đó cho đến khi bạn gọi phương thức EndDoc. EndDoc báo cho trình quản lý in là "Tôi đã hoàn thành quá trình gửi dữ liệu xuất ra cho bạn, bây giờ bạn có thể in."

Không có phương thức EndDoc, Windows sẽ tập hợp tất cả dữ liệu xuất của trình ứng dụng và không in bất kỳ dữ liệu xuất nào cho đến khi dừng ứng dụng. Nếu bạn viết một trình ứng dụng mà người dùng chạy suốt ngày và in các hóa đơn khi khách hàng mua hàng, bạn cần thực hiện phương thức EndDoc ở cuối thủ tục in từng hóa đơn nếu bạn muốn từng hóa đơn sẽ in lúc đó.

Tóm tắt

Ví dụ 21.3 sẽ in một thông báo ra đối tượng Printer và sau đó báo hiệu cho trình quản lý in rằng dữ liệu xuất đã sẵn sàng in ra giấy. Không có phương thức EndDoc, trình quản lý in sẽ giữ dữ liệu xuất ra cho đến khi trình ứng dụng có chứa mã lệnh dừng.

Củng cố

Phương thức EndDoc báo cho trình quản lý in chuyển tất cả dữ liệu xuất và in nó.

Ví dụ 21.3. *Phương thức EndDoc báo cho trình quản lý in chuyển dữ liệu in ra.*

```
1: Printer.Print "Invoice #"; invNum
2: Printer.Print "Customer:"; cust(CCnt); Tab(20); "Final Sales"
3: Printer.Print "Amount of sale:"; Tab(20); Format$(SaleAmt, "Currency")
4: Printer.Print "Tax:"; Tab(20); Format$(tax, "Currency")
5: Printer.Print
6: Printer.Print "Total:"; Tab(20); Format$(TotalSale, "Currency")
```

```
' Release the job for actual printing
Printer.EndDoc
```

Phân tích

Chương trình bao gồm mã lệnh của Ví dụ 21.3 có thể tiếp tục chạy và xử lý các tập hợp dữ liệu khác. Phương thức EndDoc được kích hoạt đảm bảo dữ liệu xuất ra được xây dựng trước phương thức Print sẽ được gửi đến máy in vật lý ngay lập tức. Nếu các phương thức Print khác xuất hiện sau đó trong chương trình, trình quản lý in sẽ bắt đầu xây dựng lại dữ liệu xuất ra một lần nữa. Quá trình gửi dữ liệu xuất kế tiếp chỉ diễn ra khi thủ tục EndDoc được gọi hay khi trình ứng dụng kết thúc.

DẤU PHÂN TRANG

Khái niệm

Khi tài liệu in đến máy in, bạn phải cẩn thận in ở đầu một trang mới khi muốn dữ liệu xuất ra trên một trang. Phương thức NewPage sẽ ép máy in nhảy trang hiện hành và bắt đầu in dữ liệu tiếp theo trên trang mới kế tiếp.

Trình quản lý in trong Windows sẽ bảo đảm rằng mỗi trang được phân trang đúng vào cuối trang vật lý. Vì vậy, nếu chiều dài trang của máy in là 66 dòng và bạn in 67 dòng, dòng 67 sẽ xuất hiện ở trang thứ 2 khi in ra.

Tuy nhiên, đôi khi bạn cần in ít hơn một trang đầy đủ trên máy in. Bạn có thể kết thúc trang in không đầy đủ đó bằng phương thức NewPage (Xem Bảng 21.1). Để dùng phương thức NewPage, đơn giản hãy áp dụng phương thức NewPage với đối tượng Printer như sau:

```
printer.NewPage
```

Ghi chú

Hãy nhớ rằng trên thực tế các phương thức xuất dữ liệu của trình ứng dụng không điều khiển trực tiếp máy in. Vì vậy, phương thức NewPage báo cho trình quản lý in biết để chuyển sang một trang mới khi trình quản lý in gặp vị trí đó trong dữ liệu xuất ra.

Nên nhớ rằng bạn đang làm việc với các máy in hỗ trợ nhiều phông chữ và kích cỡ phông chữ. Bạn có thể luôn luôn xác định số dòng in ra cho vừa một trang bằng cách kiểm tra trước tiên giá trị công thức sau:

```
numLinesPerPage = Printer.Height / Printer.TextHeight("X")
```

Như đã giải thích trong Bảng 21.1, thuộc tính Height xác định độ cao tính theo twip của trang. Thuộc tính TextHeight xác định độ cao đầy đủ của ký tự được in (bao gồm cả vùng ảnh hưởng leading, là vùng trực tiếp ở trên và dưới ký tự). TextHeight cho biết độ cao theo đơn vị twip nếu bạn không thay đổi tỉ lệ trong thuộc tính ScaleMode.

Để in các báo cáo, bạn ít khi sử dụng phương thức ScaleMode. Tuy nhiên, nếu cần thay đổi đơn vị đo, bạn phải thay đổi tỉ lệ trở lại đơn vị twip trước khi tính toán số dòng in ra trên trang như sau:

```
printer.ScaleMode = TWIPS
```

ScaleMode chấp nhận các giá trị được định nghĩa trong Bảng 21.3. Ngay khi bạn thêm tập tin CONSTANT.BAS vào cửa sổ Property của trình ứng dụng, bạn có thể dùng các tên hằng thay cho giá trị số nếu muốn thay đổi đơn vị đo tỉ lệ.

Bảng 21.3. *Các giá trị* ScaleMode.

Giá trị	Tên hằng	Diễn giải
0	USER	Giá trị do người dùng định nghĩa.
1	TWIPS	Đơn vị đo twip (mặc định).
2	POINTS	Đơn vị đo point.
3	PIXELS	Đơn vị đo pixel (đơn vị nhỏ nhất có thể định địa chỉ với máy in).
4	CHARACTERS	Đơn vị đo ký tự (120 x 240 twip).
5	INCHES	Đơn vị đo inch.
6	MILLIMETERS	Đơn vị đo mm.
7	CENTIMETERS	Đơn vị đo cm.

Tóm tắt

Ví dụ 21.4 trình bày mã lệnh in hai thông báo, mỗi thông báo trên một trang.

Củng cố

Tại điểm bất kỳ trong suốt quá trình in của trình ứng dụng, bạn có thể thực hiện phương thức NewPage để bắt máy in nhảy trang hiện hành và bắt đầu quá trình in ở đầu trang kế tiếp.

Ví dụ 21.4. *Phương thức NewPage ép máy in nhảy qua trang hiện hành.*

```
1: Printer.Print "The Report begins on the next page..."
2: Printer.NewPage ' Go to top of new page
3: Printer.Print "The Campaign Platform"
```

Phân tích

Dòng 2 nhảy trang máy in ngay cả khi máy in chưa in xong một trang đầy đủ.

Bài tập

Kiến thức tổng quát

1. Visual Basic không hỗ trợ lệnh tập tin nào?
2. Trình quản lý in của Windows thực hiện công việc gì trong quá trình xuất kết quả từ trình ứng dụng của bạn?
3. Tại sao chương trình của bạn không cần biết các lệnh riêng biệt của từng máy in?
4. Online (trực tuyến) có nghĩa là gì?
5. Làm sao bạn có thể đảm bảo máy in đã chuẩn bị trước khi bạn in các báo cáo từ trình Visual Basic?
6. Đối tượng Printer là gì?
7. Đúng hay Sai: Đối tượng Printer hỗ trợ các thuộc tính và phương thức như nhiều điều khiển khác.
8. *Pixel* là gì?
9. Phương thức Printer được dùng phổ biến nhất là gì?
10. Đúng hay Sai: Bạn có thể in các biến và hằng nhưng không in được các biểu thức với lệnh Print.

11. Đúng hay Sai: Không có điểm khác biệt giữa việc sử dụng nhiều hàm Tab(14) được nhúng với việc sử dụng dấu phẩy để phân cách các cột kết quả xuất ra trong phương thức Print.
12. Quá trình in 3 chuỗi được phân cách bởi dấu phẩy sẽ tạo nên các chuỗi xuất hiện ở cột 1, 15, và 29. Tại sao bạn không nghĩ rằng quá trình in 3 số dương, được phân cách bằng dấu phẩy, làm cho các số được in ở cột 2, 16, và 30?
13. Vùng in *(Print zone)* là gì?
14. Đúng hay Sai: Phương thức Print không kết thúc với dấu ; ở cuối luôn in ký tự trở về đầu dòng và xuống dòng trên máy in.
15. Bạn có thể in các dấu nháy kép trên giấy bằng cách nào?
16. Đúng hay Sai: Nếu việc cuối cùng một trình ứng dụng thực hiện trước khi dừng là in một báo cáo, không cần xác định phương thức EndDoc ở cuối quá trình in.

Tìm lỗi kỹ thuật

17. An Huy không cách nào in được một bản báo cáo hoàn chỉnh. Anh ấy vẫn biết một trang in có 66 dòng, ấy vậy mà chẳng thể nào in được. Hãy cho An Huy biết anh ấy phải làm thế nào để khắc phục vấn đề.
18. An Huy đang thao tác trên hai chương trình. Vì lý do nào đó, anh ấy không hình dung ra được tại sao những dữ liệu xuất ra khác nhau của hai chương trình này lại không tạo cùng một kết quả. Giúp anh ấy hóa giải vấn đề đi.

```
Printer.Print Tab(10); "Visual Basic"
Printer.Print Spc(10); "Visual Basic"
```

Kết quả xuất ra là gì ?

19. Kết quả xuất ra từ hai phương thức Print sau?

```
Printer.Print "Line 1";
Printer.Print "Line 2"
```

20. Phương thức Print sau đây sẽ xuất ra điều gì trên máy in khi trình quản lý in thực hiện in ?

```
Printer.Print "The Spanish N is "; Chr$(164)
```

Lập trình...

21. Viết một dòng lệnh Visual Basic in từ America ở cột 57.

Phần nâng cao

Viết một chương trình in các ký tự ASCII (chỉ in những giá trị ASCII từ 32 đến 255) trên giấy khi người dùng nhấn một nút lệnh.

Bài 22

Đồ họa ảo

- Điều khiển đường kẻ (Line) và hình dạng (Shape)
- Làm chủ điều khiển đường kẻ
- Làm chủ điều khiển hình dạng
- Nếu có các tập tin đồ họa...

Đa số lập trình viên Visual Basic thích học và sử dụng những công cụ trình bày trong bài này. Bài này khảo sát tỉ mỉ các thành phần điều khiển đồ họa mà một lập trình viên Visual Basic có thể sử dụng.

Có lẽ bạn sẽ không vẽ những bức ảnh đẹp cho từng mẫu biểu bạn đang thiết kế, nhưng biết cách vẽ đoạn thẳng và đường tròn sẽ cho phép bạn làm gọn mẫu biểu bằng cách phân chia các thành phần điều khiển và nhãn tương ứng trong các khung ê–lip màu để hiện sáng thông tin quan trọng.

ĐIỀU KHIỂN ĐƯỜNG KẺ (LINE) VÀ HÌNH DẠNG (SHAPE)

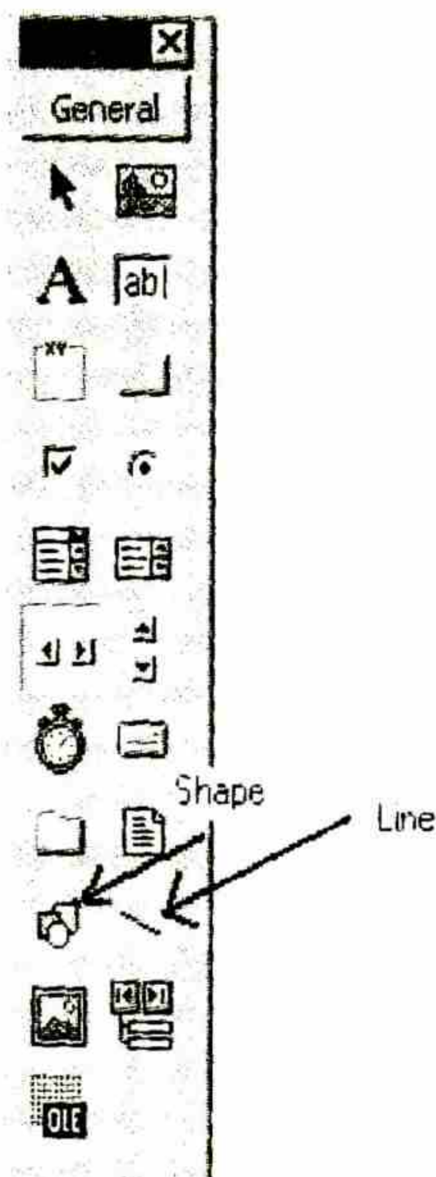
Khái niệm

Điều khiển đường kẻ và hình dạng kết hợp với nhau để vẽ đường, hộp và các loại hình tròn trên mẫu biểu. Bằng cách đặt các điều khiển trên mẫu biểu và thiết đặt những thuộc tính thích hợp, bạn sẽ làm cho trình ứng dụng tinh tế hơn.

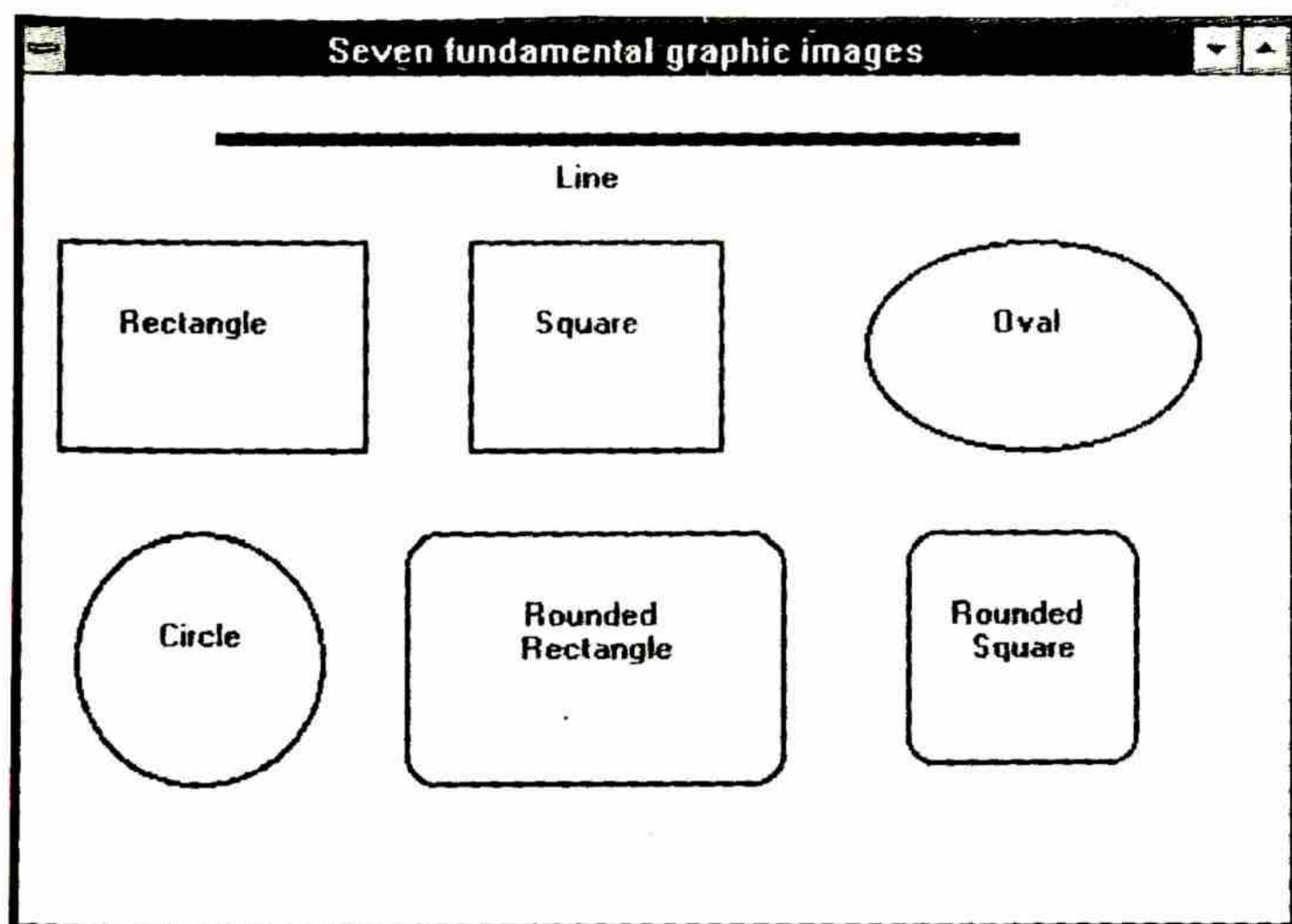
Hình 22.1 mô tả vị trí điều khiển đường kẻ và hình dạng trên cửa sổ Toolbox. Thuộc tính của từng điều khiển bạn đặt trên mẫu biểu sẽ xác định chính xác loại hình ảnh nào mà điều khiển có thể chứa.

Sau đây là một số hình ảnh đồ họa chính có thể vẽ với điều khiển đường kẻ và hình dạng:

- Đường thẳng
- Hình chữ nhật
- Hình vuông
- Hình ê–lip
- Hình tròn
- Hình chữ nhật tròn góc
- Hình vuông tròn góc



Hình 22.1. *Điều khiển đường kẻ và hình dạng cung cấp các công cụ nghệ thuật trong Visual Basic.*



Hình 22.2. *Bảy hình cơ bản có thể vẽ.*

Hình 22.2 mô tả bảy hình này. Bằng cách kết hợp các hình *ảnh hình học* cơ bản và thiết đặt những thuộc tính màu và kích cỡ thích hợp, bạn có thể vẽ hầu như bất kỳ điều gì cần trên mẫu biểu.

Sử dụng điều khiển line để vẽ các đường có độ rộng, chiều dài và những mẫu hình khác nhau. Điều khiển shape xử lý quá trình vẽ các hình dạng cơ bản khác.

Củng cố

Điều khiển đường kẻ (Line) và hình dạng (Shape) là các điều khiển vẽ chính. Có bảy hình ảnh hình học cơ bản có thể vẽ. Bằng cách xác định các thuộc tính khác nhau, bạn có thể điều khiển cách thức những hình dạng này xuất hiện trên mẫu biểu.

LÀM CHỦ ĐIỀU KHIỂN ĐƯỜNG KẺ

Khái niệm

Điều khiển đường kẻ có các thuộc tính xác định độ rộng và chiều dài những đường bạn vẽ. Thêm vào đó, bạn có thể thay đổi kiểu đường vẽ.

Bảng 22.1 liệt kê các giá trị thuộc tính của điều khiển line. Thuộc tính BorderStyle đòi hỏi bảng giải thích riêng. Bảng 22.2 chứa các giá trị có thể xác định cho BorderStyle. Thuộc tính BorderStyle xác định kiểu đường kẻ Visual Basic sẽ sử dụng. Bằng cách xác định các giá trị BorderStyle khác nhau, bạn có thể thay đổi kiểu đường kẻ. Nếu gán thuộc tính BorderStyle lúc chạy chương trình, hoặc bạn có thể gán một số cho thuộc tính BorderStyle, hoặc dùng một tên hằng được định nghĩa trong tập tin CONSTANT.BAS.

Bảng 22.1. *Các thuộc tính điều khiển đường kẻ.*

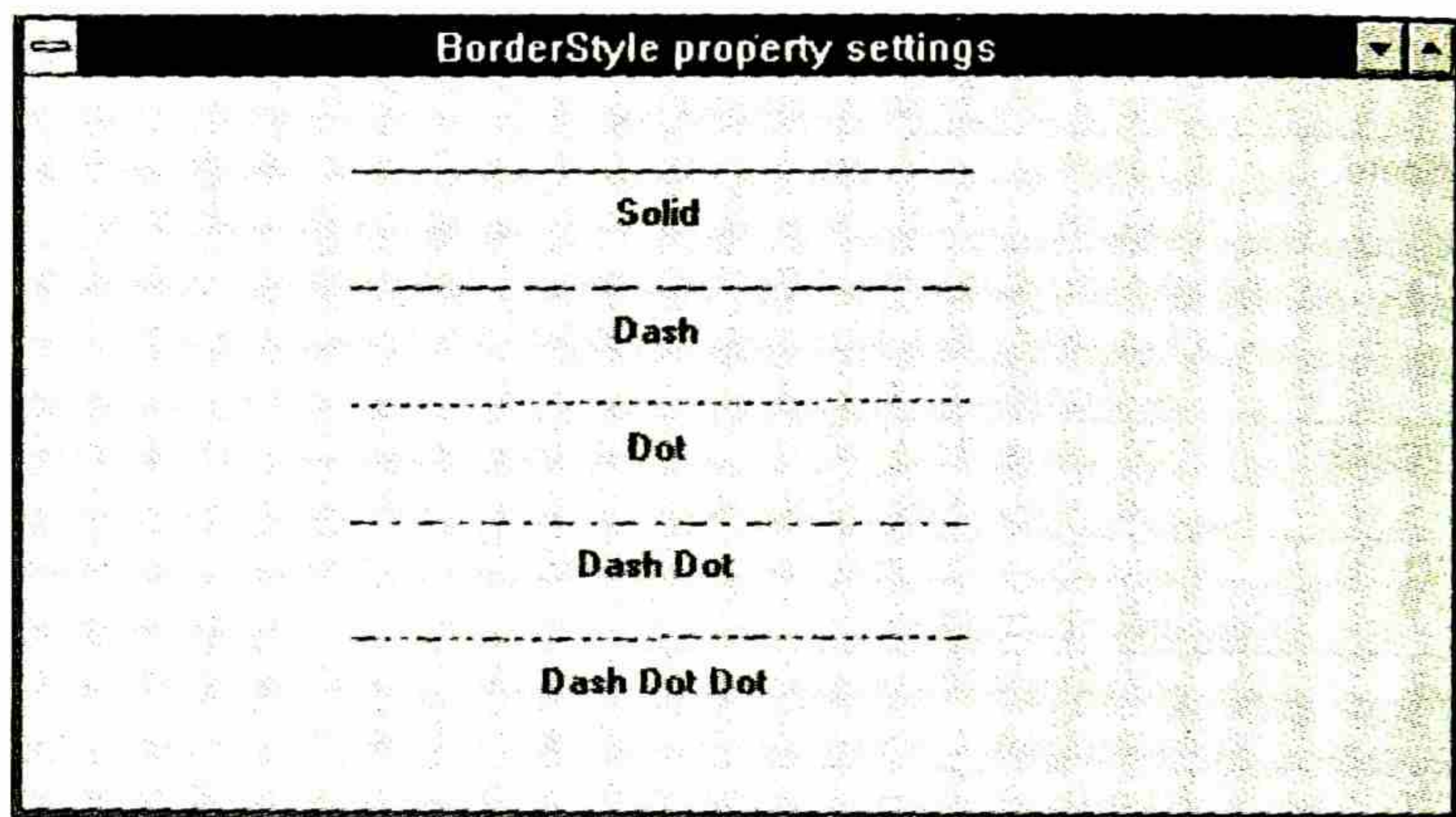
Thuộc tính	Mô tả
BorderColor	Xác định giá trị màu đường kẻ trong Windows theo hệ thập lục phân.
BorderStyle	Có thể có một trong bảy giá trị xác định kiểu đường kẻ. Hãy xem Bảng 22.2 để biết tác dụng của những giá trị này. Giá trị mặc định là 1–Solid. Thuộc tính BorderStyle không có tác dụng với các đường có thuộc tính BorderWidth lớn hơn 1 twip.

BorderWidth	Xác định kích cỡ độ rộng đường kẻ tính theo twip.
DrawMode	Một kiểu cao cấp xác định cách các mẩu bit của đường kẻ sẽ tác động đến dáng vẻ bit của mẫu biểu bao quanh. Giá trị mặc định là 13 – Copy Pen, làm việc tốt với hầu hết các trình ứng dụng Visual Basic.
Index	Xác định chỉ số điều khiển đường kẻ nếu điều khiển là một thành phần của một mảng điều khiển.
Name	Tên điều khiển line. Visual Basic đặt tên các điều khiển line là Line1, Line2, và ..., trừ khi bạn đổi tên các điều khiển line.
Tag	Không được Visual Basic sử dụng. Thuộc tính này dành cho lập trình viên xác định chú thích được áp dụng với điều khiển đường kẻ (Line).
Visible	Có giá trị True hay False, cho biết liệu người dùng có thể thấy điều khiển đường kẻ.
X1	Khoảng cách twip từ bên trái cửa sổ Form đến điểm bắt đầu đường kẻ.
X2	Khoảng cách twip từ bên trái cửa sổ Form đến điểm cuối đường kẻ.
Y1	Khoảng cách twip từ cạnh trên cùng cửa sổ Form đến điểm bắt đầu bên trái đường.
Y2	Khoảng cách twip từ cạnh trên cùng cửa sổ Form đến điểm dưới cùng ở cuối đường.

Bảng 22.2. *Các giá trị thuộc tính BorderStyle của điều khiển đường kẻ.*

Giá trị	Tên CONSTANT.BAS	Mô tả
0–Transparent	TRANSPARENT	Có thể thấy nền mẫu biểu xuyên qua đường kẻ.
1–Solid	SOLID	Đường kẻ liên tục.
2–Dash	DASH	Đường kẻ dạng gạch nối.
3–Dot	DOT	Đường kẻ dạng điểm.
4–Dash–Dot	DASH_DOT	Đường kẻ có dạng dãy gạch nối kế tiếp là các điểm.
5–Dash–Dot–Dot	DASH_DOT_DOT	Đường kẻ có dạng một gạch nối kế tiếp là 2 điểm.
6–Inside Solid	INSIDE_SOLID	Giống như đường kẻ 1–Solid.

Hình 22.3 minh họa cách các xác lập thuộc tính BorderStyle khác nhau tác động lên đường bạn vẽ. Thuộc tính BorderStyle xác định cách phân bố số gạch nối và điểm trong mẫu đường kẻ. (Điều này giống như chúng ta đang nói về mã lệnh tín hiệu Morse.)



Hình 22.3. *Tác động của thuộc tính BorderStyle.*

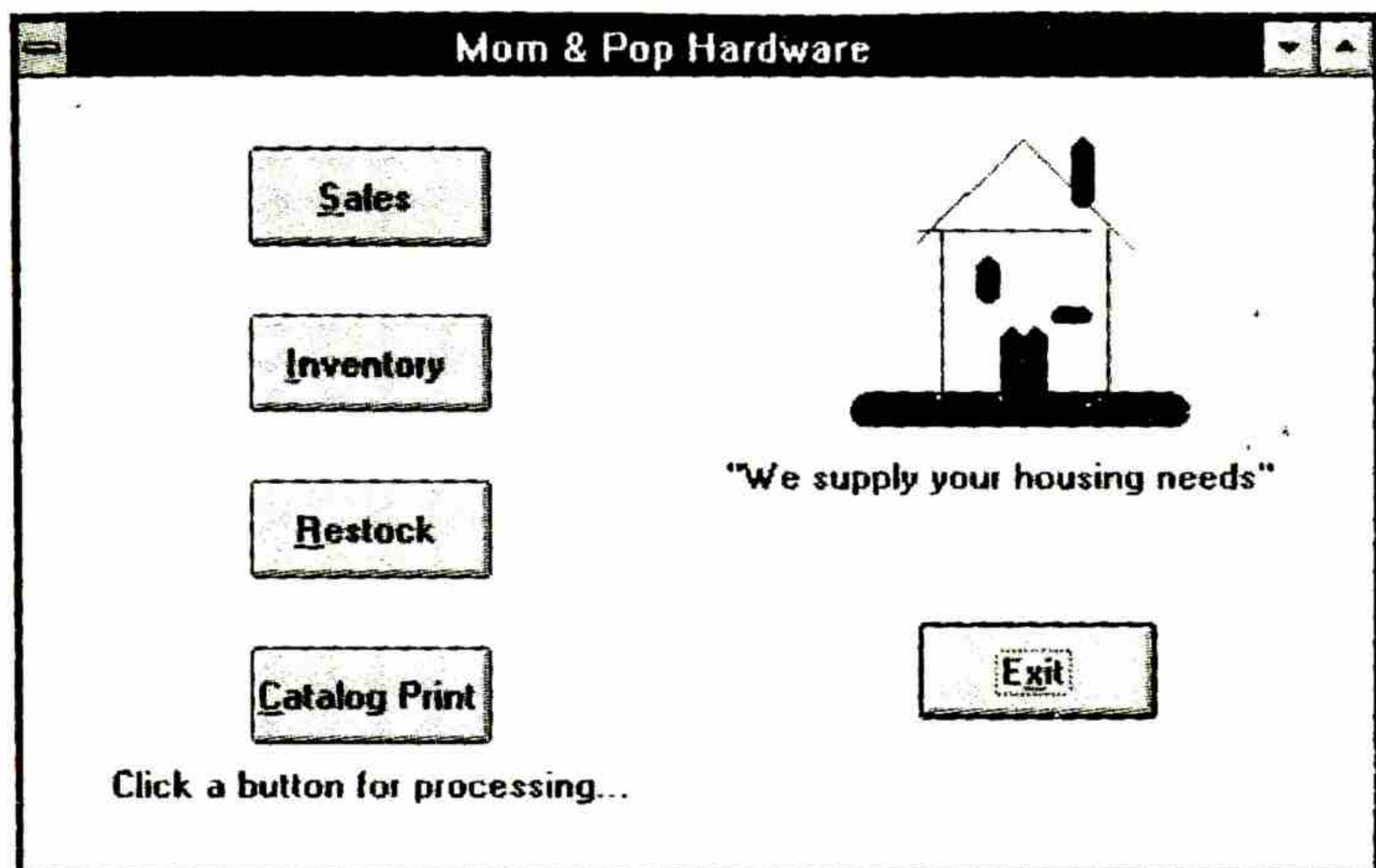
Để vẽ một đường, nhấp đúp điều khiển đường kẻ trên hộp công cụ. Một đường kẻ xuất hiện ở giữa mẫu biểu với hai điều khiển ở hai đầu đường kẻ. Để di chuyển đường kẻ tới vị trí khác, kéo điểm giữa đường kẻ bằng mouse. Để thay đổi chiều dài đường kẻ, kéo một trong hai điều khiển trên đường. Bạn có thể kéo lên hay xuống điểm cuối đường kẻ bằng cách kéo điều khiển ở cuối đường bằng mouse.

Mách nước

Sau khi định vị trí đường bằng mouse gần đúng với vị trí bạn cần đường xuất hiện, có thể điều chỉnh chính xác kích cỡ và vị trí đường bằng cách thiết đặt các giá trị thuộc tính khác nhau. Nếu kiên nhẫn, bạn có thể làm sinh động đường kẻ bằng cách thay đổi các xác lập thuộc tính X1, X2, Y1, và Y2 lặp đi lặp lại thông qua mã lệnh.

Tóm tắt

Hình 22.4 mô tả cửa sổ Form lưu một màn hình có vẻ chuyên nghiệp. Toàn bộ ngôi nhà được vẽ bằng cách dùng một dãy các đường kẻ.



Hình 22.4. *Các đường kẻ tạo thêm vẻ đẹp quyến rũ cho một mẫu biểu.*

Lưu ý

Hàm duy nhất đã nối với mẫu biểu này là lệnh End bên trong thủ tục biến cố cmdExit_Click() của nút lệnh Exit.

Củng cố

Bằng cách dùng điều khiển hình kẻ và xác định các giá trị thuộc tính khác nhau, bạn có thể vẽ các hình dạng trên nền của mẫu biểu.

LÀM CHỦ ĐIỀU KHIỂN TẠO HÌNH

Khái niệm

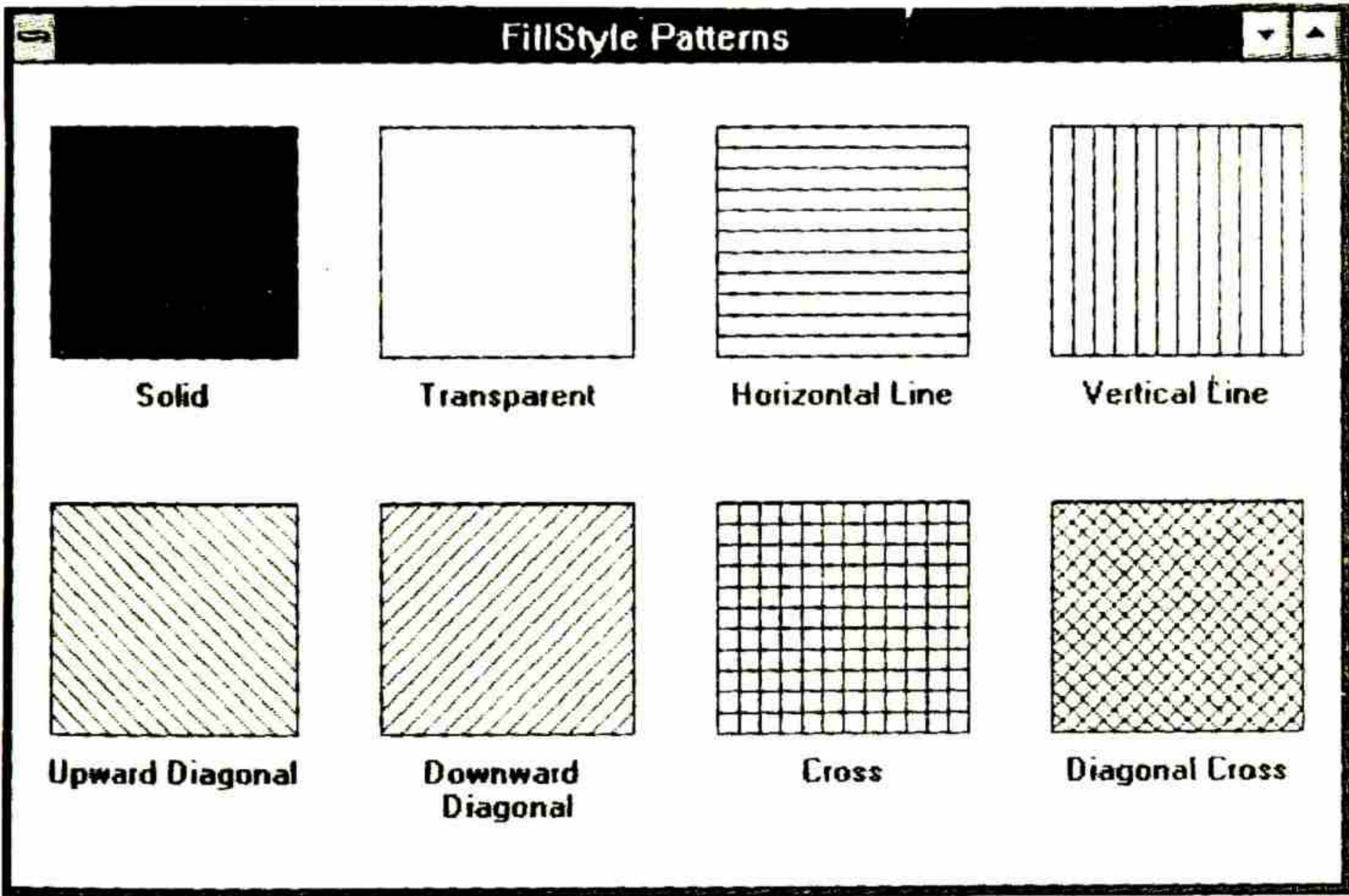
Điều khiển hình dạng cho bạn khả năng vẽ sáu loại hình ảnh khác nhau trên mẫu biểu. Các thuộc tính màu và hình dạng khác nhau giúp bạn phân biệt dạng hình này với dạng hình khác.

Bảng 22.3 bao gồm các thuộc tính có hiệu lực cho điều khiển hình dạng (Shape). Thuộc tính quan trọng nhất là thuộc tính Shape cung cấp một trong sáu hình dạng cơ bản.

Bảng 22.3. *Các thuộc tính điều khiển hình dạng.*

Thuộc tính	Mô tả
BackColor	Xác định giá trị màu nền của hình theo hệ thập lục phân.
BackStyle	Bao gồm hoặc giá trị 0–Transparent (mặc định) hoặc 1–Opaque xác định liệu nền mẫu biểu có thể thấy xuyên qua hình hay sẽ bị che.
BorderColor	Xác định màu viền của hình.
BorderStyle	Gồm bảy giá trị xác định mẫu viền của hình. Các giá trị BorderStyle của điều khiển hình dạng trong Bảng 22.2 cung cấp các giá trị BorderStyle có thể có của hình. Giá trị mặc định là 1–Solid. Thuộc tính BorderStyle không có hiệu lực trên các hình ảnh khi thuộc tính BorderWidth lớn hơn 1 twip.
BorderWidth	Xác định kích cỡ twip của đường viền quanh hình.
DrawMode	Một kiểu cao cấp xác định cách các mẫu bit của hình tác động dạng bit của mẫu biểu bao quanh. Giá trị mặc định là 13 – Copy Pen, hoạt động tốt với tất cả trình ứng dụng Visual Basic.
FillColor	Xác định màu tô bên trong hình với giá trị theo hệ thập lục phân.
FillStyle	Gồm 8 giá trị xác định kiểu đường Visual Basic vẽ bên trong hình. Bảng 22.4 trình bày các giá trị có thể có cho thuộc tính FillStyle của hình. Giá trị mặc định FillStyle là 0–Solid.
Height	Xác định độ cao theo twip của hình (từ điểm cao nhất đến điểm thấp nhất của hình).
Index	Xác định chỉ số điều khiển hình dạng nếu điều khiển hình dạng là một thành phần của mảng điều khiển.
Left	Xác định khoảng cách twip từ biên trái mẫu biểu đến biên trái của hình.
Name	Tên của điều khiển hình dạng (Shape). Visual Basic đặt tên cho các điều khiển Shape là Shape1, Shape2, và v.v, trừ phi bạn đổi tên chúng.

Shape	Gồm một trong sáu giá trị xác định kiểu hình cho điều khiển hình dạng. Bảng 22.5 gồm các giá trị có thể có cho thuộc tính Shape của hình. Thuộc tính Shape mặc định là 0–Rectangle.
Tag	Không được Visual Basic sử dụng, thuộc tính này dành cho lập trình viên xác định chú thích về điều khiển Shape được áp dụng.
Top	Xác định khoảng cách twip từ biên đầu mẫu biểu đến đỉnh cao nhất của hình.
Visible	Có giá trị True hoặc False, cho biết liệu người dùng có thể nhìn thấy điều khiển hình dạng.
Width	Xác định độ rộng twip của hình (nơi rộng nhất).



Hình 22.5. *Thuộc tính FillStyle xác định nền của dạng hình sẽ hiển thị.*

Bảng 22.4 gồm các giá trị có thể có về thuộc tính FillStyle của điều khiển shape. Hình 22.5 mô tả các mẫu tô khác nhau mà hình có thể chứa.

Bảng 22.4. *Các giá trị FillStyle của điều khiển Shape*

Giá trị	Tên hằng trong CONSTANT.BAS	Mô tả
0–Solid	SOLID	Tô đầy mẫu.
1–Transparent	TRANSPARENT	Hình chỉ thấy đường viền.
2–Horizontal Line	HORIZONTAL_LINE	Tô hình với các đường ngang.
3–Vertical Line	VERTICAL_LINE	Tô hình với các đường dọc.
4–Upward Diagonal	UPWARD_DIAGONAL	Tô hình với các đường chéo hướng lên.
5–Downward Diagonal	DOWNWARD_DIAGONAL	Tô hình với các đường chéo hướng xuống.
6–Cross	CROSS	Tô hình dạng carô.
7–Diagonal Cross	DIAGONAL_CROSS	Tô hình với các đường chéo carô.

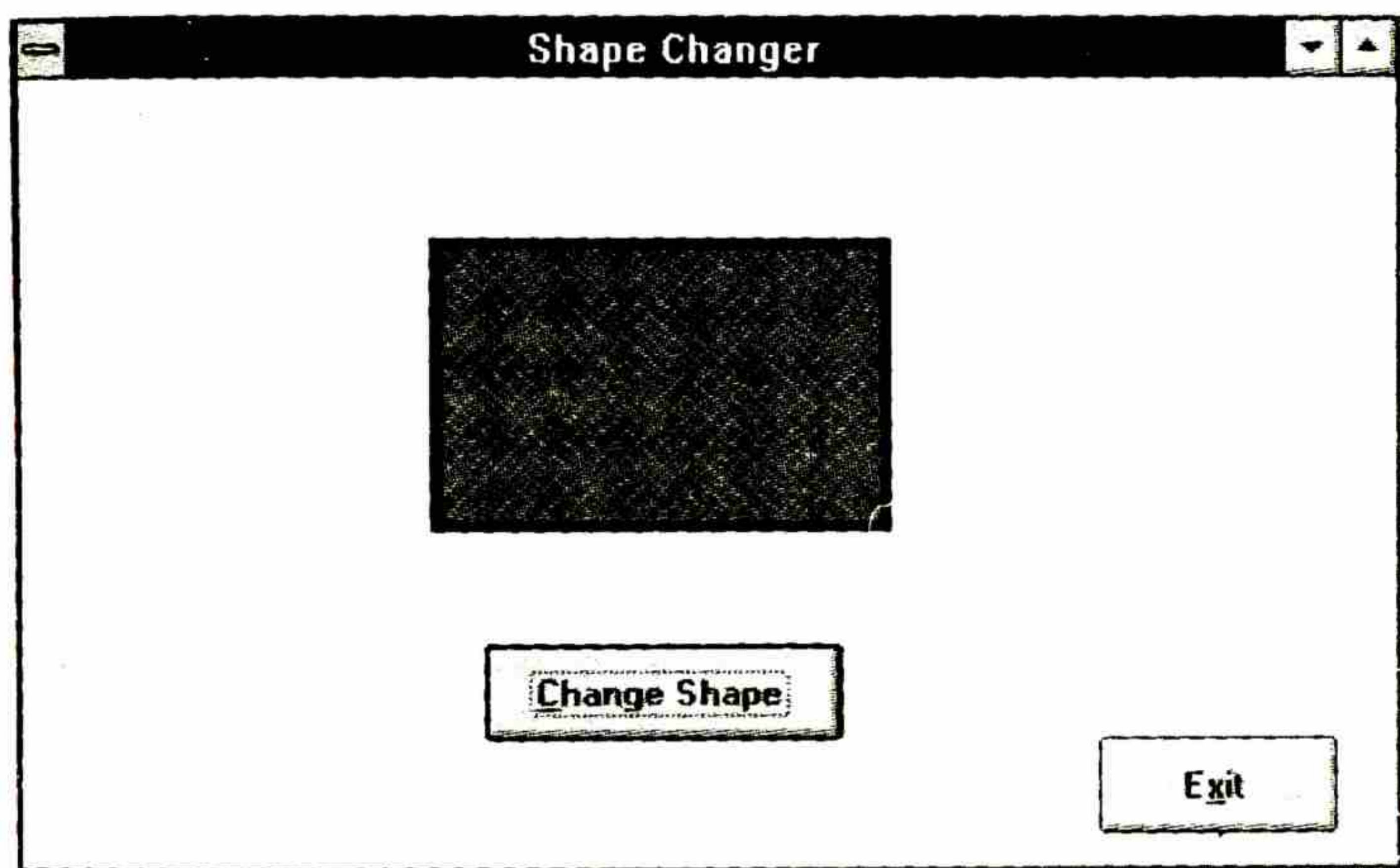
Bảng 22.5 gồm các giá trị có thể có của thuộc tính Shape trong điều khiển hình dạng (Shape). Hình 22.2 minh họa các dạng hình khác nhau mà điều khiển hình dạng có thể chứa. Vì vậy, khi muốn đặt một hình vuông lên mẫu biểu, bạn sẽ đặt điều khiển hình dạng lên mẫu biểu và định thuộc tính Shape bằng 1–Square.

Bảng 22.5. *Các thuộc tính Shape của điều khiển hình dạng.*

Giá trị	Tên hằng trong CONSTANT.BAS	Mô tả
0–Rectangle	SHAPE_RECTANGLE	Hình chữ nhật
1–Square	SHAPE_SQUARE	Hình vuông
2–Oval	SHAPE_OVAL	Hình ê–lip
3–Circle	SHAPE_CIRCLE	Hình tròn
4–Rounded Rectangle	SHAPE_ROUNDED_RECTANGLE	Hình chữ nhật tròn góc
5–Rounded Square	SHAPE_ROUNDED_SQUARE	Hình vuông tròn góc

Tóm tắt

Hình 22.6 cung cấp một trình ứng dụng hình đơn giản sẽ thay đổi hình dạng khi người dùng nhấn nút lệnh. Bạn có thể nạp và chạy ứng dụng SHAPECH.VBP.



Hình 22.6. *Thay đổi hình bằng cách nhấp một nút.*

Củng cố

Điều khiển hình dạng cung cấp 6 trong số 7 hình cơ bản có thể đặt trên mẫu biểu. Thuộc tính Shape xác định đường viền của hình.

NẾU CÓ CÁC TẬP TIN ĐỒ HỌA...

Khái niệm

Visual Basic hỗ trợ hai điều khiển đồ họa không phải để vẽ nhưng bạn có thể đặt các biểu tượng và ảnh nhị phân mà bạn có trên đĩa.

Hình 22.7 minh họa vị trí hai điều khiển đồ họa đó được gọi là điều khiển Picture Box và điều khiển Image. Hai điều khiển này hầu như y hệt nhau và cho phép nạp các ảnh trên đĩa vào mẫu biểu của trình ứng dụng.



Hình 22.7. *Điều khiển Image và Picture Box trên thanh công cụ.*

Lưu ý

Điều khiển Picture Box cung cấp một vài đặc tính cao cấp mà bạn có thể dùng nếu viết trình ứng dụng MDI *(giao diện nhiều tài liệu). Điều khiển Image hiệu quả hơn và hiển thị ảnh nhanh hơn điều khiển Picture Box.*

Sau đây là ba loại tập tin đồ họa mà các điều khiển này có thể nạp từ đĩa và hiển thị trên mẫu biểu:

- Tập tin Bitmap với phần mở rộng tên tập tin là .BMP hay .DIB
- Tập tin Icon với phần mở rộng tên tập tin là .ICO
- Tập tin Metafiles với phần mở rộng tên tập tin là .WMF

Bạn có thể lấy các tập tin đồ họa này ở đâu? Có nhiều nguồn như vậy. Nếu sử dụng Microsoft Paintbrush trong Windows, bạn có thể tạo các tập tin Bitmap. Tập tin Icon là tập tin đồ họa nén chứa biểu tượng trong chương trình Windows. Nhiều trình ứng dụng Windows khác lưu ảnh đồ họa dưới dạng metafile.

Bảng 22.6 mô tả các thuộc tính chung và quan trọng có hiệu lực cho cả hai điều khiển Image và Picture Box. Giá trị quan trọng nhất là thuộc tính Picture chứa tên tập tin và đường dẫn tới hình sẽ được hiển thị bên trong điều khiển.

Bảng 22.6. *Các thuộc tính của điều khiển Picture Box và Image*

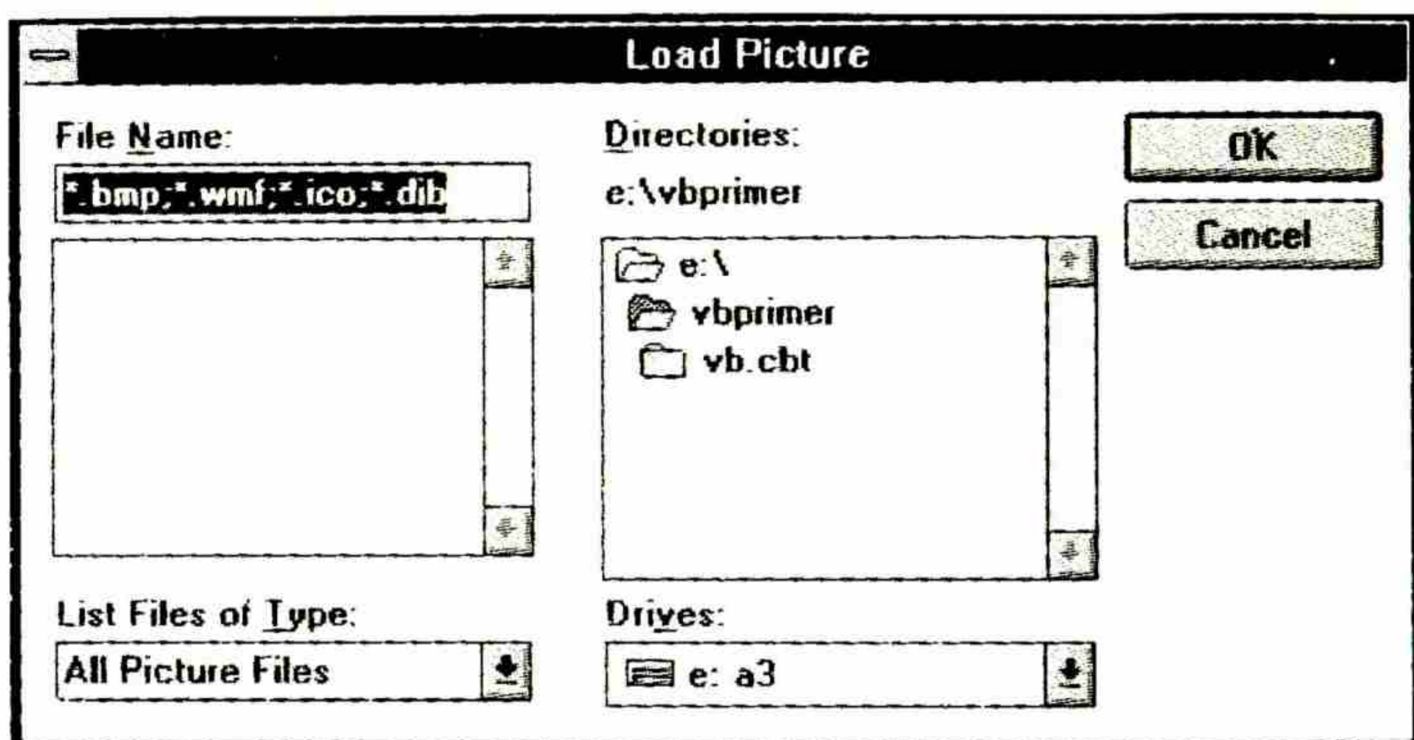
Thuộc tính	Mô tả
BorderStyle	Mô tả kiểu đường viền quanh hình. Nếu định là 0–None (mặc định), không có đường viền quanh hình. Nếu là 1–Fixed Single, một đường viền sậm sẽ bao quanh hình.
Height	Chiều cao của hình tính theo twip. (Xem thuộc tính Stretch.)
Left	Khoảng cách twip từ cạnh trái cửa sổ Form tới cạnh trái của hình.
Name	Tên điều khiển Picture Box hay Image. Visual Basic tự đặt tên các điều khiển Picture Box là Picture1, Picture2, ..., và tên điều khiển Image là Image1, Image2, ... trừ khi bạn đổi tên các điều khiển.
Picture	Đường dẫn và tên tập tin đầy đủ của hình ảnh Visual Basic sẽ hiển thị trong điều khiển lúc chạy chương trình. Không thấy hình trong điều khiển cho đến khi người dùng chạy chương trình.
Stretch	(Chỉ tác dụng với điều khiển Image). Nếu định là False (mặc định), điều khiển sẽ tự định lại kích cỡ cho phù hợp kích cỡ của hình. Bạn không cần xác định kích cỡ điều khiển, điều khiển sẽ tự định kích cỡ đúng với kích cỡ hình của tập tin trên đĩa. Nếu là True, hình sẽ tự định lại kích cỡ cho phù hợp với điều khiển.
Width	Độ rộng của điều khiển tính theo twip. (Xem thuộc tính Stretch.)

Khi muốn đặt một ảnh đồ họa lên mẫu biểu, bạn chỉ cần đặt điều khiển Image (hay Picture Box) lên mẫu biểu và xác định tên tập tin cho hình muốn điều khiển hiển thị. Hãy mở mẫu biểu mới và thực hiện các bước sau để tạo trình ứng dụng với một hình ảnh:

1. Nhấp đúp điều khiển Image để đặt điều khiển ở giữa mẫu biểu.
2. Nhấn F4 hiển thị cửa sổ Property.

3. Nhấp thuộc tính Picture. Bạn sẽ thấy hình ê–lip xuất hiện nơi nhập vào giá trị thuộc tính.
4. Nếu nhấp hình ê–lip, Visual Basic mở hộp thoại Load Picture như minh họa trong Hình 22.8. Trong hộp thoại, bạn xác định tập tin bất kỳ, bao gồm cả đường dẫn chỉ rõ hình muốn hiển thị trên mẫu biểu.

Thuộc tính Picture sẽ lưu tên đường dẫn và tập tin của hình bạn đặt trên mẫu biểu. Lúc chạy chương trình, Visual Basic sẽ hiển thị hình nơi đặt điều khiển.



Hình 22.8. *Hộp thoại Load Picture điền nội dung vào thuộc tính Picture.*

Bạn cũng có thể thay đổi tập tin đồ họa sẽ xuất hiện trong điều khiển Image hay Picture Box bằng cách dùng mã lệnh Visual Basic. Hàm LoadPicture sẽ nạp những tập tin đồ họa xác định vào các điều khiển đồ họa để hiển thị hình ảnh này trên mẫu biểu. Câu lệnh sau sẽ thay đổi tập tin hình ảnh dùng cho một điều khiển Image tên imgFace:

```
imgFace.Picture = LoadPicture("D:\FIGS\FACE.BMP")
```

Bạn có thể thay thế các hình trong thời gian chạy chương trình bằng cách nạp những tập tin khác nhau bằng hàm LoadPicture().

Mách nước

Bằng cách dùng một chuỗi rỗng, "", cho đối số hàm LoadPicture(), bạn có thể xóa hình trong điều khiển Picture Box hay Image.

Củng cố

Phần này thảo luận về khả năng phối hợp đồ họa trong chương trình Visual Basic. Không chỉ Visual Basic cung cấp những công cụ cần thiết để vẽ hình riêng, mà còn cho phép nạp hình ảnh từ các tập tin trên đĩa vào mẫu biểu.

Bài tập

Kiến thức tổng quát

1. Hai điều khiển nào cho phép bạn vẽ hình ảnh hình học trên mẫu biểu?
2. Có bao nhiêu loại bóng khác nhau do điều khiển hình dạng (Shape) cung cấp?
3. Giá trị thuộc tính nào xác định mẫu các đường kẻ?
4. Thuộc tính nào xác định đối tượng được vẽ bằng điều khiển hình dạng?
5. Hai điều khiển nào cho phép nạp các tập tin hình ảnh đồ họa vào mẫu biểu?
6. Điều khiển hình ảnh đồ họa nào là hiệu quả nhất trong hai điều khiển?
7. Ba loại tập tin đồ họa mà Visual Basic có thể hiển thị trên mẫu biểu?
8. Thuộc tính nào buộc hình ảnh đồ họa co lại trong điều khiển Picture Box hay Image?
9. Thuộc tính điều khiển nào xác định đường dẫn và tên tập tin của hình ảnh đồ họa để hiển thị?
10. Tên hàm Visual Basic nạp tập tin hình ảnh đồ họa lúc chạy chương trình là gì?

Tìm lỗi kỹ thuật

11. An Huy đã vẽ một đường dày 15 twip. Anh muốn dùng đường kẻ để phân biệt một thông báo mẫu biểu lớn với các điều khiển còn lại trên mẫu biểu. Vấn đề là An Huy dường như không thể thực hiện được việc anh ấy muốn là đường kẻ dày phải hiển thị như một chuỗi gạch nối. Visual Basic từ chối đáp ứng thuộc tính BorderStyle của anh ấy và luôn hiển thị một đường liên tục. Hãy giải thích cho An Huy biết vấn đề anh ấy gặp phải.
12. Bạn có thể nghĩ ra cách khác để hoàn thành đường kẻ dày với các gạch nối của An Huy?

Kết quả xuất ra là gì?

13. An Huy đã viết một chương trình có lệnh sau:

```
imgPainting.Picture = LoadPicture("")
```

Điều gì sẽ xuất hiện trên điều khiển Image khi trình ứng dụng của An Huy gọi hàm LoadPicture()?

Viết mã lệnh...

14. Giả sử bạn muốn vẽ một hình chữ nhật có đường viền xanh da trời, đường chéo màu đỏ và nền xanh lá cây. Hãy mô tả các thuộc tính của điều khiển hình dạng (Shape) mà bạn phải ấn định màu xanh da trời, đỏ và xanh lá cây.

Phần nâng cao

Viết chương trình vẽ một khuôn mặt màu xanh lá cây lớn ở giữa mẫu biểu. Thêm một nút lệnh mà người dùng có thể nhấp để cặp mắt trên khuôn mặt hạnh phúc đó nhấp nháy.

Bài thực hành 11

Làm đẹp chương trình

Tóm tắt

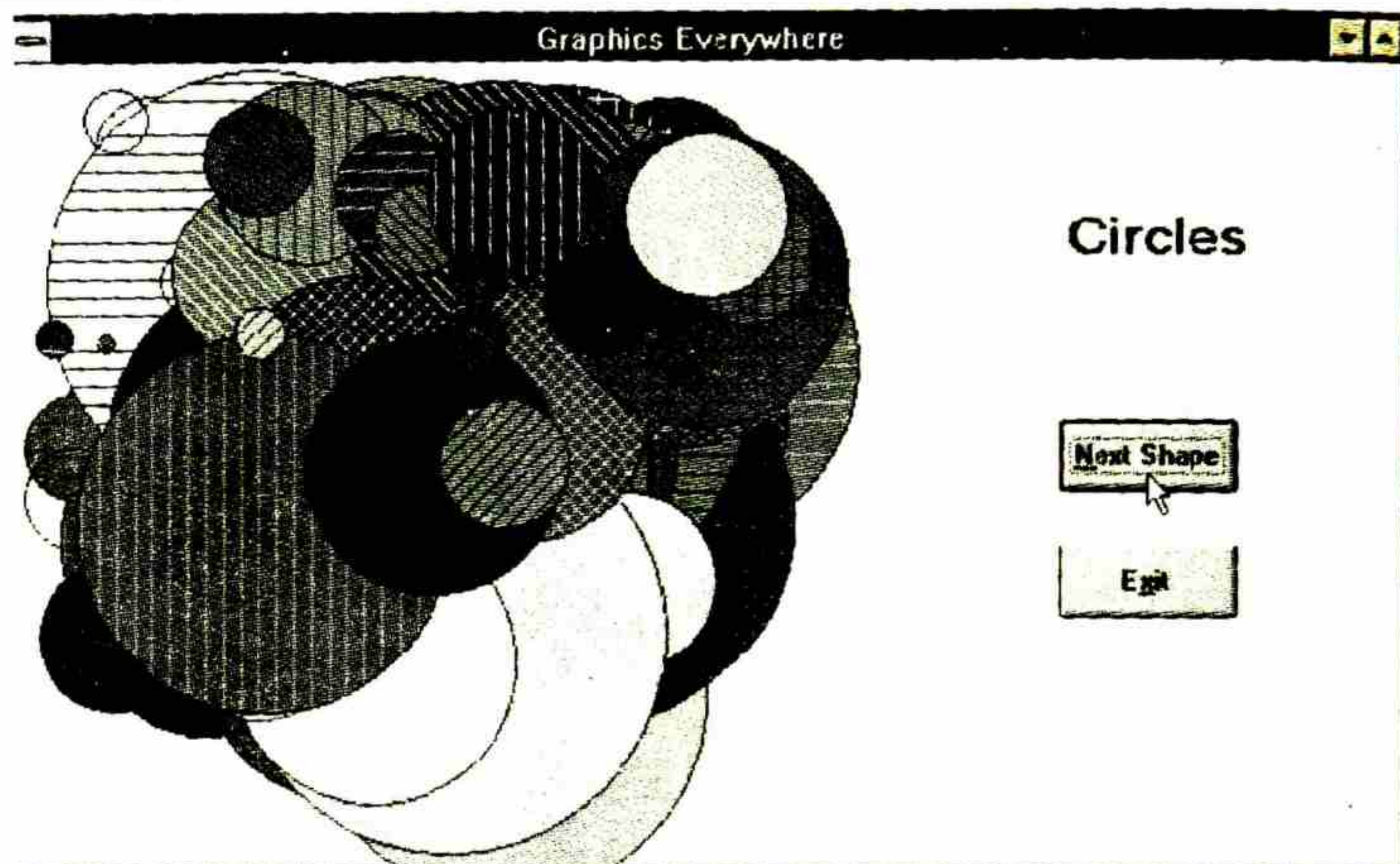
Chương này chỉ cho bạn cách in các báo cáo văn bản ra máy in và cách trang trí màn hình với hình ảnh. Từ "có thể thấy được – *visual*" trong Visual Basic trở nên quá rõ ràng khi bạn thêm hình ảnh tạo chú ý của người dùng. Có lẽ điều khiển hình dạng (Shape) cung cấp nhiều hình dạng khác nhau hơn bất kỳ điều khiển khác, cho phép bạn vẽ hình vuông, hình tròn, hình oval theo nhiều dạng, kích cỡ và màu sắc. Bài thực hành này lấy điều khiển hình dạng để minh họa quá trình trang trí mẫu biểu cho bắt mắt với hơn 150 hình tròn, hình vuông và hình oval có tô màu.

Trong chương này, bạn đã tìm hiểu:

- Khi nào sẽ truy xuất đối tượng Printer
- Cách sử dụng phương thức Print để xuất văn bản trên trang in
- Những hình dạng nào có hiệu lực với điều khiển hình dạng
- Cách vẽ các đường trên mẫu biểu bằng cách dùng điều khiển Line
- Khi nào sử dụng điều khiển Picture Box và điều khiển Image để đặt các tập tin đồ họa trên mẫu biểu

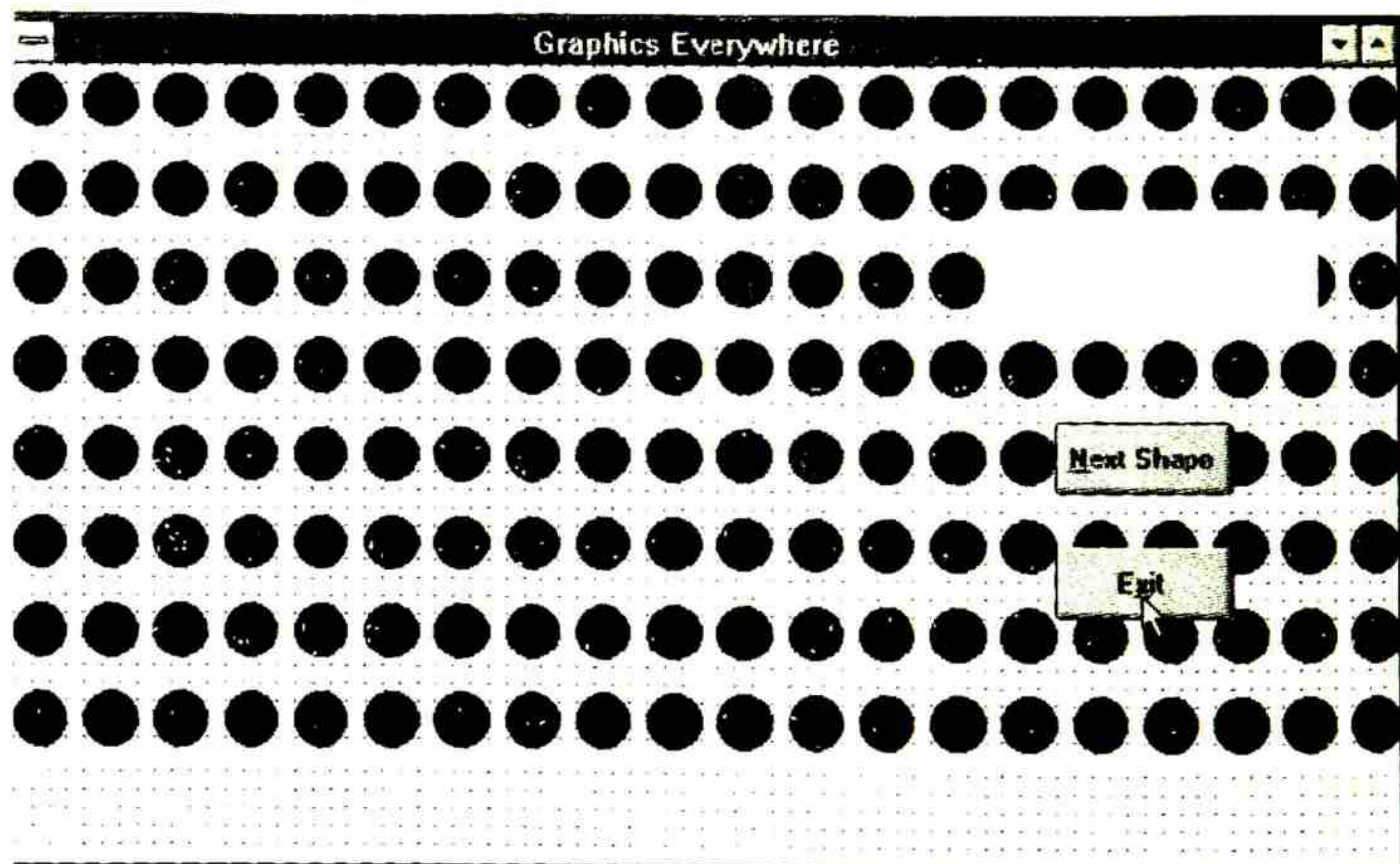
Mô tả chương trình

Hình P11.1 minh họa cửa sổ Form đang mở của PROJEC11.VBP! Khi bạn nạp và thi hành chương trình, nó sẽ phát sinh tất cả loại hình tròn trên mẫu biểu. Toàn bộ tập hợp hình tròn, cũng như tập hợp hình kế tiếp sẽ xuất hiện sau hình tròn, tất cả đều khác nhau nhờ vào hàm Rnd() trong chương trình, đã được giới thiệu trong Bài Thực Hành 10.



Hình P11.1. *Project11 vẽ các đường tròn trên trình ứng dụng khác.*

Khi bạn nhấp nút lệnh Next Shape, chương trình sẽ thay thế đường tròn với các hình vuông tròn góc. Nhấn Next Shape một lần nữa sẽ hiển thị một dãy hình oval. Bạn có thể tiếp tục hiển thị lại các hình tròn, vuông, và oval nếu muốn cho đến khi nhấn nút lệnh Exit để kết thúc chương trình.



Hình P11.2. *Chế độ thiết kế của Project11 minh họa nơi đặt mảng điều khiển chứa các hình.*

Chương trình gồm một mảng điều khiển với 160 điều khiển hình tên shpShape. Hình P11.2 minh họa mẫu biểu lúc thiết kế. Vị trí điều khiển hình lúc thiết kế không giống như lúc thi hành chương trình bởi vì mã lệnh dùng hàm Rnd() để hiển thị các hình tại những vị trí ngẫu nhiên trên màn hình, với màu sắc, kích cỡ và mẫu tô ngẫu nhiên.

Chương trình phụ thuộc vào hàm Rnd() là hàm đặc biệt lấy một số ngẫu nhiên từ 1 đến giá trị đối số bất kỳ được chuyển tới hàm. Tên của thủ tục Function là RndNum(); bạn sẽ thấy hàm này trong mô tả mã lệnh xuất hiện trong Ví dụ P11.1.

Ví dụ P11.1. *Mã lệnh phát sinh tất cả hình ảnh trên mẫu biểu.*

```
Option Explicit

Sub Form_Activate ()
' Get things started by displaying circles
Call Circles
End Sub

Sub Circles ()
' Draws several circles of different
' colors and sizes on the form
Dim Count As Integer
lblShape.Caption = "Circles"
For Count = 0 To 159 ' 160 total shapes
shpShape(Count).Shape = SHAPE_CIRCLE
shpShape(Count).Visible = False
shpShape(Count).Top = RndNum(3000)
shpShape(Count).Width = RndNum(5000)
shpShape(Count).Height = RndNum(3000)
shpShape(Count).Left = RndNum(3000)
shpShape(Count).BackColor = QBColor(RndNum(16) – 1) ' Adjust for zero
shpShape(Count).FillColor = QBColor(RndNum(16) – 1) ' Adjust for zero
shpShape(Count).Visible = True
shpShape(Count).FillStyle = RndNum(8) – 1 ' Adjust for zero
Next Count
```

```
End Sub

Sub Ovals ()
' Draws several ovals of different
' colors and sizes on the form
Dim Count As Integer
lblShape.Caption = "Ovals"
For Count = 0 To 159 ' 160 total shapes
shpShape(Count).Shape = SHAPE_OVAL
shpShape(Count).Visible = False
shpShape(Count).Top = RndNum(3000)
shpShape(Count).Width = RndNum(4000)
shpShape(Count).Height = RndNum(3000)
shpShape(Count).Left = RndNum(3000)
shpShape(Count).BackColor = QBColor(RndNum(16) – 1) ' Adjust for zero
shpShape(Count).FillColor = QBColor(RndNum(16) – 1) ' Adjust for zero
shpShape(Count).Visible = True
shpShape(Count).FillStyle = RndNum(8) – 1 ' Adjust for zero
Next Count
End Sub

Sub Squares ()
' Draws several squares of different
' colors and sizes on the form
Dim Count As Integer
lblShape.Caption = "Squares"
For Count = 0 To 159 ' 160 total shapes
shpShape(Count).Shape = SHAPE_SQUARE
shpShape(Count).Visible = False
shpShape(Count).Top = RndNum(3000)
shpShape(Count).Width = RndNum(5000)
shpShape(Count).Height = RndNum(3000)
shpShape(Count).Left = RndNum(3000)
shpShape(Count).BackColor = QBColor(RndNum(16) – 1) ' Adjust for zero
shpShape(Count).FillColor = QBColor(RndNum(16) – 1) ' Adjust for zero
```

```
60: shpShape(Count).Visible = True
61: shpShape(Count).FillStyle = RndNum(8) - 1 ' Adjust for zero
62: Next Count
63: End Sub
64:
65: Sub cmdShape_Click ()
66: ' Check the caption of the description
67: ' label to see what shapes to draw next
68: Select Case Left$(lblShape.Caption, 1)
69: Case "C":
70: Call Squares
71: Case "S"
72: Call Ovals
73: Case "O":
74: Call Circles
75: End Select
76: End Sub
77:
78: Sub cmdExit_Click ()
79: End
80: End Sub
81:
82: Function RndNum (n)
83: ' Returns a random number from 1 to n
84: RndNum = Int(n * Rnd + 1)
85: End Function
```

Mô tả

1: Yêu cầu tất cả các biến phải được định nghĩa trước khi sử dụng.

2: Dòng trống giúp phân tách từng phần chương trình.

3: Mã lệnh thủ tục biến cố sẽ thi hành ngay khi người dùng thấy mẫu biểu lần đầu tiên.

4: Chú thích giải thích thủ tục.

(4: Luôn nhớ khởi tạo mẫu biểu trước tiên.)

5: Thi hành thủ tục Subroutine vẽ hình tròn.

6: Kết thúc thủ tục.

7: Dòng trống phân tách các thủ tục.

8: Thủ tục Subroutine vẽ hình tròn.

9: Chú thích giải thích thủ tục.

10: Chú thích tiếp theo.

11: Định nghĩa biến sẽ điều khiển vòng lặp.

12: Đặt đề mục trên nhãn mô tả.

13: Duyệt qua tất cả 160 thành phần của mảng điều khiển.

(13: Có 160 thành phần trong mảng điều khiển Shape để hiển thị.)

14: Định hình tròn cho từng điều khiển Shape.

15: Giấu điều khiển tạm thời cho đến khi các thuộc tính được thiết đặt.

16: Định vị trí thích hợp ở đầu mẫu biểu.

17: Định độ rộng thích hợp.

18: Định độ cao thích hợp.

19: Định vị trí thích hợp so với góc trái mẫu biểu.

20: Định màu nền bằng cách gọi hàm QBColor() có sẵn và chuyển vào một giá trị từ 0 đến 15.

21: Định màu tô bằng cách gọi hàm QBColor() có sẵn và chuyển vào một giá trị từ 0 đến 15.

22: Hiển thị điều khiển với thuộc tính kích cỡ và vị trí được ấn định.

23: Tô hình với màu tô có số từ 0 đến 8.

24: Chuẩn bị hình kế tiếp trong mảng điều khiển.

25: Kết thúc thủ tục.

26: Dòng trống phân tách các thủ tục.

27: Thủ tục Subroutine vẽ các hình oval.

28: Chú thích giải thích thủ tục.

29: Chú thích tiếp theo.

30: Định nghĩa biến sẽ điều khiển vòng lặp.

31: Đặt đề mục cho nhãn mô tả.

32: Duyệt qua 160 thành phần trong mảng điều khiển.

33: Đặt hình oval cho từng điều khiển hình dạng.

34: Giấu điều khiển tạm thời cho đến khi các thuộc tính được thiết đặt.

35: Định một vị trí thích hợp ở đầu mẫu biểu.

36: Định độ rộng thích hợp.

37: Định chiều cao thích hợp.

38: Định vị trí thích hợp ở góc trái mẫu biểu.

39: Định màu nền bằng cách gọi hàm QBColor() có sẵn và chuyển một giá trị từ 0 đến 15.

(39: Hàm QBColor() nhận một giá trị từ 0 đến 15 trình bày các màu khác nhau.)

40: Định màu tô bằng cách gọi hàm QBColor() có sẵn và chuyển vào một giá trị từ 0 đến 15.

41: Hiển thị điều khiển với thuộc tính vị trí và kích cỡ đã định.

42: Tô hình với màu có số từ 0 đến 8.

43: Chuẩn bị cho hình kế tiếp trong mảng điều khiển.

44: Kết thúc thủ tục.

45: Dòng trống phân tách các thủ tục.

46: Thủ tục Subroutine vẽ hình vuông.

47: Chú thích giải thích thủ tục.

48: Chú thích tiếp theo.

49: Định nghĩa biến điều khiển vòng lặp.

50: Đặt đề mục cho nhãn mô tả.

51: Duyệt qua tất cả 160 thành phần trong mảng điều khiển.

52: Đặt hình vuông cho từng điều khiển hình dạng.

(52: Tên hằng giúp bạn khỏi phải nhớ các giá trị thuộc tính số tương ứng.)

53: Giấu điều khiển tạm thời cho đến khi tất cả thuộc tính được thiết đặt.

54: Định vị trí thích hợp ở đầu mẫu biểu.

55: Định độ rộng thích hợp.

56: Định chiều cao thích hợp.

57: Định vị trí thích hợp ở góc trái mẫu biểu.

58: Định màu nền bằng cách gọi hàm QBColor() có sẵn và chuyển vào một giá trị từ 0 đến 15.

59: Định màu tô bằng cách gọi hàm QBColor() có sẵn và chuyển vào một giá trị từ 0 đến 15.

60: Hiển thị điều khiển với thuộc tính vị trí và kích cỡ đã định.

61: Tô hình với màu tô có giá trị từ 0 đến 8.

62: Chuẩn bị hình kế tiếp trong mảng điều khiển.

63: Kết thúc thủ tục.

64: Dòng trống phân tách các thủ tục.

65: Thủ tục thi hành để hiển thị tập hợp các hình kế tiếp khi người dùng nhấp nút lệnh cmdShape.

66: Chú thích giải thích thủ tục.

67: Chú thích tiếp theo.

68: Xác định hình kế tiếp bằng cách nhìn nhãn mô tả hình dạng được hiển thị trước đó.

(68: Mã lệnh tìm ký tự đầu tiên của nhãn để quyết định hình dạng nào sẽ được hiển thị tiếp theo.)

69: Nếu nhãn chứa nội dung Circles...

70: Hiển thị các hình vuông ngẫu nhiên.

71: Nếu nhãn có nội dung là Squares...

72: Hiển thị các hình oval ngẫu nhiên.

73: Nếu nhãn có nội dung là Ovals...

74: Hiển thị các hình tròn ngẫu nhiên.

75: Kết thúc lệnh Select.

76: Kết thúc thủ tục.

77: Dòng trống phân tách các thủ tục.

78: Thủ tục kết thúc chương trình khi người dùng nhấp nút lệnh cmdExit.

79: Kết thúc quá trình thi hành chương trình.

80: Kết thúc thủ tục.

81: Dòng trống phân tách các thủ tục.

82: Thủ tục Function nhận một đối số nguyên và trả lại một số ngẫu nhiên trong khoảng từ 1 đến giá trị đối số đó.

83: Chú thích giải thích hàm.

84: Định giá trị trả lại của hàm để phát sinh số ngẫu nhiên.

(84: Thủ tục Function đệ qui giúp bạn lặp mã lệnh hàm một số lần xác định.)

85: Kết thúc thủ tục.

Đóng trình ứng dụng

Bây giờ bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Chương tiếp theo sẽ chỉ bạn cách dùng các điều khiển còn lại và công cụ dò tìm lỗi trực tuyến của Visual Basic để trợ giúp tìm kiếm và sửa lỗi trong mã lệnh.

Chương XII

Bài 23

Thanh cuốn, khung lưới và mouse

- Cuộn các thanh cuốn
- Chuẩn bị điều khiển khung lưới
- Sử dụng điều khiển khung lưới
- Giám sát con trỏ mouse
- Chặn quá trình nhấp và di chuyển mouse

Bài này trình bày các điều khiển còn lại trong Visual Basic giúp bạn cách truy xuất điều khiển khung lưới (Grid) và thanh cuốn (Scroll Bar) đặc biệt, cũng như nắm được quá trình di chuyển và nhấp mouse của người dùng. Như tất cả điều khiển khác, quá trình tìm hiểu cách làm việc của thanh cuốn và khung lưới đòi hỏi bạn phải làm chủ các thuộc tính liên quan đến điều khiển đó.

Ghi chú

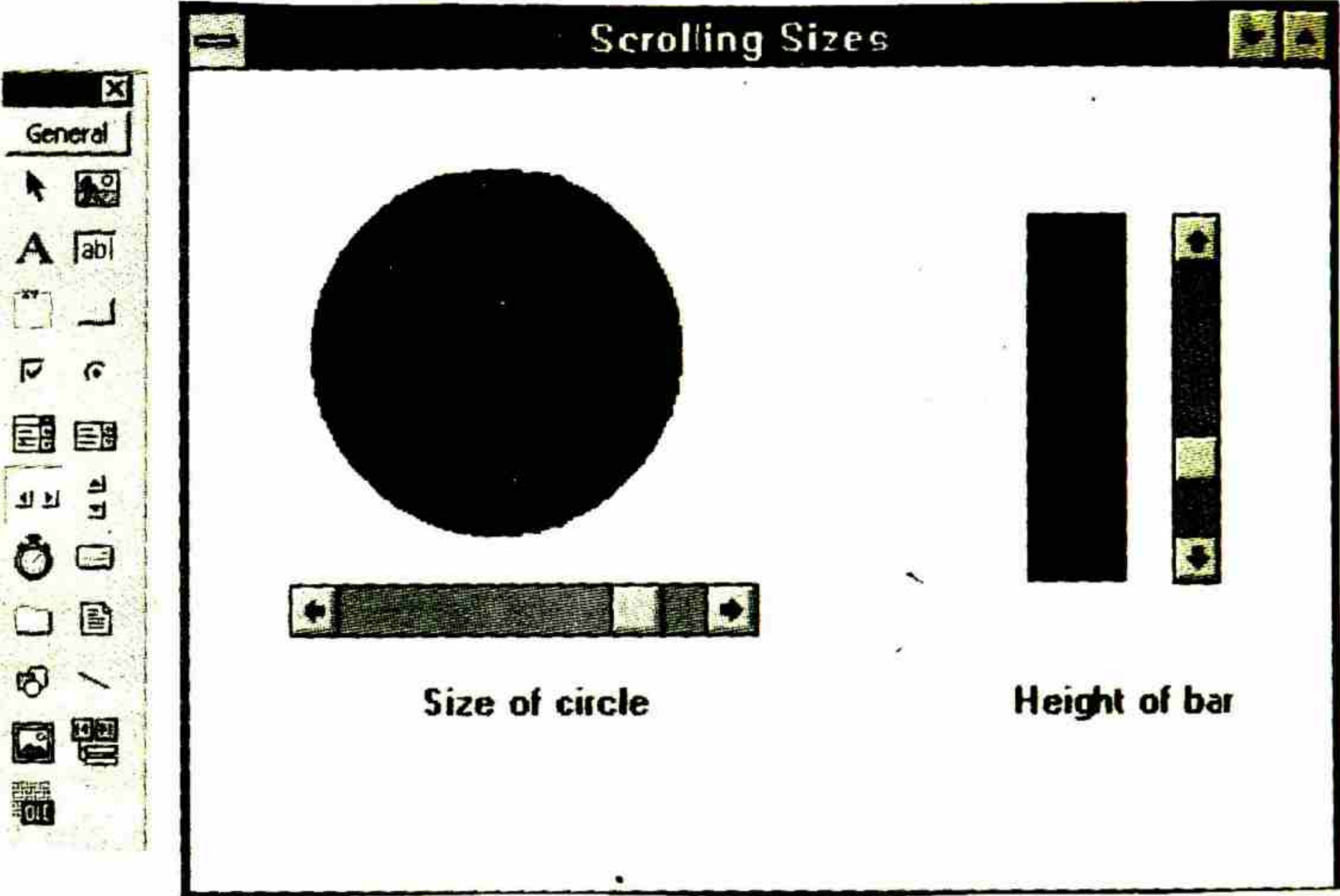
Điều khiển khung lưới (Grid) đòi hỏi chuẩn bị vài thứ. Ban đầu, điều khiển khung lưới không xuất hiện trên hộp công cụ cho đến khi bạn thêm điều khiển đó vào.

CUỘN CÁC THANH CUỐN

Khái niệm

Thanh cuốn hỗ trợ cho người dùng khả năng kiểm soát quá trình thay đổi dữ liệu. Thay vì nhập giá trị xác định, người dùng có thể di chuyển thanh cuốn bằng mouse để xác định vị trí của một giá trị tương ứng trong dãy.

Hình 23.1 mô tả vị trí của hai điều khiển thanh cuốn (Scroll Bar) trên cửa sổ Toolbox. Đó là thanh cuốn ngang và thanh cuốn dọc. Ngoài ra, bạn có thể thấy hình dạng hai thanh cuốn trên mẫu biểu cho phép người dùng điều chỉnh thuộc tính chiều rộng và chiều cao của nó bằng cách nhấp mouse trên thanh cuốn đó.



Hình 23.1a. *Vị trí và hình dạng các thanh cuốn trên cửa sổ Toolbox.*

Hình 23.1b. *Minh họa về các thanh cuốn.*

Bảng 23.1 liệt kê danh sách các thuộc tính thanh cuốn. Thuộc tính quan trọng nhất và duy nhất cho riêng thanh cuốn là LargeChange, Max, Min, và SmallChange.

Bảng 23.1. *Các thuộc tính thanh cuốn*

Thuộc tính	Mô tả
DragIcon	Xác định biểu tượng sẽ xuất hiện khi người dùng kéo thanh cuốn trên mẫu biểu. (Bạn ít khi cho phép người dùng di chuyển thanh cuốn, cho nên các xác lập thuộc tính Drag... thường không thích hợp.)

DragMode	Giá trị 1 cho yêu cầu kéo mouse bằng tay (người dùng có thể nhấn–giữ nút mouse trong khi kéo điều khiển) hoặc 0 (mặc định) cho quá trình kéo mouse tự động, nghĩa là người dùng không thể kéo thanh cuốn, nhưng bạn có thể thực hiện quá trình kéo thông qua mã lệnh nếu cần.
Enabled	Nếu định là True (mặc định), thanh cuốn có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng quá trình xử lý biến cố của điều khiển đó.
Height	Cho biết chiều cao tính theo twip của thanh cuốn.
HelpContextID	Nếu bạn bổ sung trợ giúp cảm ngữ cảnh cao cấp vào trình ứng dụng, thuộc tính HelpContextID sẽ cung cấp một số xác định văn bản trợ giúp.
Index	Nếu thanh cuốn là thanh phần của điều khiển mảng, thuộc tính Index sẽ cung cấp giá trị chỉ số cho từng thanh cuốn.
LargeChange	Xác định tổng số giá trị của thanh cuốn sẽ thay đổi mỗi khi người dùng nhấp trong thân thanh cuốn.
Left	Khoảng cách tính bằng twip từ biên trái cửa sổ Form tới biên trái thanh cuốn.
Max	Cho biết giá trị đơn vị cực đại được thiết đặt cho thanh cuốn. Mặc định, vùng giá trị từ 1 đến 32767.
Min	Cho biết giá trị đơn vị tối thiểu được thiết đặt cho thanh cuốn. Mặc định, vùng giá trị từ 1 đến 32767.
MousePointer	Qui định hình dạng con trỏ mouse lúc người dùng di chuyển con trỏ mouse trên thanh cuốn. Các giá trị có thể từ 0 đến 12, chỉ các hình dạng khác nhau mà con trỏ mouse có thể có.
Name	Qui định tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên VScroll1, VScroll2, và ... (cho thanh cuốn dọc), và HScroll1, HScroll2, ... (cho thanh cuốn ngang) khi bạn lần lượt thêm các thanh cuốn vào mẫu biểu.
SmallChange	Xác định lượng giá trị thanh cuốn sẽ thay đổi khi người dùng nhấp trên mũi tên ở cuối thanh cuốn.

TabIndex	Xác định thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần bổ sung điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm (focus) bằng cách thay đổi giá trị thuộc tính TabIndex của điều khiển. Không có hai điều khiển nào trên cùng mẫu biểu lại có cùng giá trị TabIndex.
TabStop	Nếu là True, người dùng có thể nhấn Tab để di chuyển tiêu điểm (focus) tới thanh cuốn này. Nếu là False, thanh cuốn không thể nhận tiêu điểm (focus).
Tag	Không được Visual Basic sử dụng. Đây là tiện ích dành cho lập trình viên xác định chú thích của thanh cuốn.
Top	Khoảng cách twip từ cạnh trên cùng thanh cuốn đến cạnh trên cùng mẫu biểu.
Value	Lưu giá trị theo đơn vị đo hiện hành cho biết vị trí thanh cuốn.
Visible	Có giá trị True hoặc False, cho biết liệu người dùng có thể thấy và sử dụng thanh cuốn.
Width	Độ rộng twip của thanh cuốn.

Mách nước

Tên thanh cuốn ngang bắt đầu hsb và thanh cuốn dọc bằng vsb để bạn có thể dễ dàng phân biệt chúng với nhau.

Khi đặt thanh cuốn trên mẫu biểu, bạn phải quyết định giới hạn các giá trị mà thanh cuốn sẽ trình bày. Vùng giá trị đầy đủ của thanh cuốn có thể mở rộng từ 1 đến 32767. Hãy định thuộc tính Min cho giá trị sẽ trình bày thấp nhất trong thanh cuốn và thuộc tính Max cho giá trị cao nhất.

Khi người dùng sử dụng thanh cuốn, các mũi tên trên thanh cuốn sẽ điều khiển quá trình thay đổi từng khoảng giá trị nhỏ tùy vào giá trị được xác định trong thuộc tính SmallChange. Quá trình nhấp vùng trống bên trong thanh cuốn sẽ làm tăng hay giảm giá trị hiện hành của thanh cuốn đi một khoảng bằng giá trị trong thuộc tính LargeChange. Người dùng có thể tự kéo thanh cuốn tới vị trí bất kỳ trong thân thanh cuốn để nhảy nhanh đến một giá trị xác định thay vì thay đổi giá trị từ từ.

Ví dụ, giả sử thanh cuốn ngang trình bày khoản tiền đô từ $5 đến $100. Khi người dùng nhấp các mũi tên cuộn, giá trị thanh cuốn sẽ thay đổi 1 đô–la. Khi người dùng nhấp vào vùng thân trống trong hộp cuộn, giá trị thanh cuốn thay đổi 5 đô–la. Sau đây là các giá trị thuộc tính bạn phải thiết đặt để Visual Basic hiểu :

Min: 5

Max: 100

SmallChange: 1

LargeChange: 5

Kích cỡ vật lý của thanh cuốn không phụ thuộc vào giá trị trả về của thanh cuốn khi người dùng chọn. Hãy điều chỉnh các thanh cuốn trên mẫu biểu cho đủ rộng và cao thích hợp với kích cỡ các mục chúng cần trình bày.

Tóm tắt

Ví dụ 23.1 trình bày mã lệnh SCROLL.VBP bạn có thể nạp và chạy để điều chỉnh kích cỡ đường tròn và đường kẻ mà bạn đã thấy trong Hình 23.1.

Củng cố

Có hai thanh cuốn, thanh cuốn ngang và thanh cuốn dọc, cho người dùng khả năng chọn một giá trị từ vùng giá trị có thể có mà không cần nhập vào những giá trị đó.

Ví dụ 23.1. *Mã lệnh ứng dụng SCROLL.VBP.*

```
Option Explicit

Sub Form_Load ()
' Set initial scroll bar values
hsbBar.Value = 1800 ' Circle's default width
vsbBar.Value = 1800 ' Bar's default height
End Sub

Sub hsbBar_Change ()
' As user clicks the scroll bar,
```

```
' the width of the circle adjusts
shpCircle.Width = hsbBar.Value
End Sub

Sub vsbBar_Change ()
' As user clicks the scroll bar,
' the height of the bar adjusts
shpBar.Height = vsbBar.Value
End Sub
```

Phân tích

Sau đây là các thuộc tính quan trọng của thanh cuốn ngang được thiết đặt trong quá trình thiết kế mẫu biểu:

Min: 50

Max: 2100

SmallChange: 50

LargeChange: 100

Điều khiển hình dạng gồm có hình tròn (tên shpCircle) có thuộc tính Width là 2100, cho nên hình tròn lớn nhất có thể xuất hiện trong điều khiển là 2100 twip. Do đó định thuộc tính Max giá trị 2100.

Sau đây là các thuộc tính quan trọng của thanh cuốn dọc được đặt trong quá trình thiết kế mẫu biểu:

Min: 50

Max: 2300

SmallChange: 50

LargeChange: 100

Điều khiển hình dạng chứa đường kẻ (tên shpBar) có thuộc tính Height bằng 2300, cho nên đường kẻ cao nhất có thể xuất hiện trong điều khiển là 2300 twip. Do đó định thuộc tính Max giá trị 2300.

Hai thuộc tính Min đều lưu giá trị 50 cho biết hình tròn và đường kẻ được thu nhỏ lại khi người dùng giảm kích cỡ điều khiển đến mức tối thiểu.

Các dòng 5 và 6 ấn định các giá trị ban đầu cho độ rộng hình tròn và chiều cao đường kẻ là 1800, cho nên cả hai hình đều bắt đầu khá lớn. Thực tế, các dòng lệnh đặt giá trị khởi đầu cho cả hai thanh cuốn trong lúc chạy chương trình sẽ phát sinh thủ tục biến cố Change() theo sau mã lệnh.

Dòng 12 đảm bảo quá trình tăng hay giảm độ rộng hình tròn, phụ thuộc vào giá trị thuộc tính Value của thanh cuốn ngang. Dòng 18 đảm bảo quá trình tăng hay giảm chiều cao đường kẻ, phụ thuộc vào giá trị thuộc tính Value của thanh cuốn dọc.

CHUẨN BỊ ĐIỀU KHIỂN KHUNG LƯỚI

Khái niệm

Trước khi có thể dùng điều khiển khung lưới (Grid), bạn phải bổ sung điều khiển khung lưới vào cửa sổ Toolbox của trình ứng dụng.

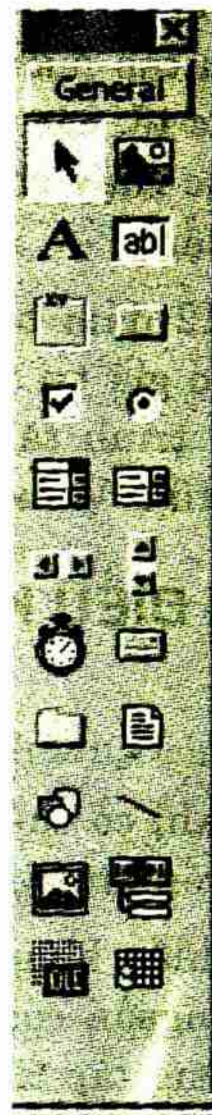
Điều khiển khung lưới là điều khiển chuyên biệt không có trong hộp công cụ nguồn của Visual Basic. Tất cả điều khiển chuyên biệt sẽ lưu trong những tập tin trên đĩa với phần mở rộng tên tập tin là OCX. Mọi ứng dụng sử dụng điều khiển khung lưới phải bổ sung tập tin DBGRID32.OCX trong cửa sổ Controls khi chọn lệnh Project Components ...

Khi không cần điều khiển khung lưới, trình ứng dụng của bạn sẽ nạp nhanh hơn và chiếm ít không gian đĩa hơn nếu gỡ bỏ tập tin DBGRID32.OCX khỏi cửa sổ Toolbox của trình ứng dụng. Cho đến nay, các bài học trong sách đều không cần điều khiển khung lưới.

Nếu muốn làm việc với điều khiển khung lưới, bạn phải thêm tập tin DBGRID32.VBX vào cửa sổ Toolbox của trình ứng dụng bất kỳ đòi hỏi điều khiển khung lưới. Bây giờ, bạn có thể quyết định có muốn thêm DBGRID32.VBX vào project riêng biệt cần sử dụng điều khiển khung lưới hay không.

Các bước sau giải thích cách thêm điều khiển khung lưới vào một project. Để bổ sung điều khiển khung lưới vào một trình ứng dụng:

1. Nạp ứng dụng .VBP.
2. Chọn lệnh Project ➡ Components... Visual Basic sẽ hiển thị hộp thoại Components.
3. Chọn mục Controls, và đánh dấu ô chọn tập tin DBGRID32.OCX..
4. Nhấn phím Enter hay nhấp nút lệnh OK để đóng hộp thoại. Visual Basic sẽ cập nhật cửa sổ Toolbox để cửa sổ bao gồm cả điều khiển khung lưới chuyên biệt như được mô tả trong Hình 23.2.
5. Cất giữ .VBP.



Hình 23.2. *Vị trí điều khiển khung lưới trên hộp công cụ.*

Củng cố

Trước khi học cách sử dụng điều khiển khung lưới (Grid), bạn phải nạp tập tin điều khiển chuyên biệt DBGRID32.VBX vào cửa sổ Project. Bằng cách nạp tập tin vào project, tất cả project kế tiếp sẽ có điều khiển khung lưới cho đến khi bạn gỡ bỏ tập tin điều khiển tùy biến.

SỬ DỤNG ĐIỀU KHIỂN KHUNG LƯỚI

Khái niệm

Điều khiển khung lưới giúp bạn hiển thị văn bản, số và hình ảnh đồ họa trong một bảng nhiều hàng và cột.

Khi bạn phải hiển thị nhiều giá trị cùng lúc, điều khiển khung lưới (Grid) là một trong những điều khiển thuận tiện nhất để sử dụng. Mặc dù nhãn chứa các thông báo dài và riêng biệt, nhưng quá trình

cuộn các hộp danh sách sẽ khá tiện lợi đối với danh sách giá trị người dùng có thể chọn. Khung lưới cho phép trình ứng dụng của bạn hiển thị dữ liệu trong một bảng 2 chiều.

Giả sử bạn muốn hiển thị số hàng vận chuyển hàng năm của từng khách hàng trong một công ty vận chuyển đã nhận chuyển hàng cho 8 khách hàng. Mỗi hành khách chuyển 5 loại hàng mỗi năm. Tổng cộng bạn cần 40 điều khiển hiển thị kết quả tính toán tổng số từng mặt hàng vận chuyển của từng khách hàng. Mỗi điều khiển ứng với kết quả tính tổng của loại mặt hàng và khách hàng khác nhau.

Hãy nghĩ về 5 điều khiển hiển thị kết quả cho 8 khách hàng đó, bạn sẽ nhận thấy một bảng gồm 8 hàng, 5 cột sẽ là nơi hoàn hảo cho quá trình đặt dữ liệu tính toán trong một ứng dụng máy tính.

Sau khi thiết kế ứng dụng Visual Basic hiển thị các giá trị, bạn có thể dễ dàng điều chỉnh bảng hay lưu các giá trị vào một tập tin trên đĩa. Khi bạn bổ sung các khách hàng, có thể dễ dàng mở rộng kích cỡ khung lưới (grid) thông qua mã lệnh chương trình.

Khái niệm mới

Một ô là vùng giao giữa hàng với cột trong khung lưới.

Bạn hãy dành một ít thời gian nghiên cứu Hình 23.3. Hình bố trí các ô trên khung lưới có thể xuất hiện trên mẫu biểu. Kích cỡ khung lưới được xác định bởi các thuộc tính grid, mà bạn có thể thiết đặt hay điều chỉnh bằng cách dùng mouse khi đặt điều khiển khung lưới trên mẫu biểu.

Hình 23.3. *Điều khiển khung lưới đem lại một bảng nhiều hàng và cột.*

Mách nước

Nếu kích cỡ khung lưới không đủ lớn để hiển thị tất cả dữ liệu của nó, Visual Basic sẽ bổ sung các thanh cuốn để người dùng có thể cuộn trên khung lưới.

Hàng và cột được tô bóng có thể hiểu như là những hàng và cột cố định. Các xác lập giá trị thuộc tính của khung lưới sẽ xác định liệu bạn muốn đặt riêng 1 hàng hay cột cố định. Khi bạn yêu cầu một hay nhiều hàng, cột cố định (hay một tổ hợp hàng và cột), Visual Basic sẽ giữ các hàng và cột này lại trong quá trình cuộn khi người dùng cuộn các giá trị khung lưới.

Khi xem các giá trị trong khung lưới, người dùng có thể cuộn khung lưới bằng cách chọn (bằng mouse hay bàn phím) một hay nhiều ô trong khung lưới. Thông qua lập trình, bạn có thể cập nhật hay sao chép các giá trị ô được chọn đến điều khiển khung lưới (Grid) khác, đến các biến hay tập tin dữ liệu.

Bảng 23.2 trình bày các giá trị thuộc tính có hiệu lực với một điều khiển khung lưới. Mặc dù bạn đã biết nhiều thuộc tính từ các điều khiển khác, khung lưới còn có những thuộc tính riêng xác định chiều của khung lưới cũng như số cột và hàng cố định.

Bảng 23.2. *Các thuộc tính của điều khiển khung lưới*

Thuộc tính	Mô tả
About...	Nhấp thuộc tính sẽ mở hộp thoại hiển thị thông tin về điều khiển khung lưới. Đa số các điều khiển chuyên biệt đều dùng hộp thoại About này để trình bày thông tin bản quyền của điều khiển đó. Thuộc tính About chỉ có hiệu lực trong quá trình thiết kế chương trình và không làm gì lúc chạy.
BackColor	Xác định màu nền của điều khiển khung lưới, được chọn như một mã màu thập lục phân hay từ bảng màu. Thuộc tính BackColor mô tả màu ô thuộc các hàng và cột không cố định. Tất cả ô thuộc hàng và cột cố định đều có màu xám.
BorderStyle	Xác định đường viền quanh khung lưới nếu bằng 1–Fixed Single (mặc định). Không có đường viền nếu định bằng 0–None.

Cols	Số cột trong khung lưới.
DragIcon	Chứa các biểu tượng sẽ xuất hiện khi người dùng kéo khung lưới trên mẫu biểu. (Bạn ít khi cho phép người dùng di chuyển khung lưới, cho nên thuộc tính Drag... thường không thích hợp.)
DragMode	Bằng 1 tương ứng với yêu cầu kéo mouse bằng tay (người dùng có thể giữ nút mouse trong khi kéo điều khiển) hoặc bằng 0 (mặc định) cho quá trình kéo mouse tự động, nghĩa là người dùng không thể kéo khung lưới nhưng bạn có thể khởi tạo quá trình kéo nếu cần thông qua mã lệnh.
Enabled	Xác định liệu khung lưới có thể đáp ứng các biến cố. Nếu định là True (mặc định), khung lưới có thể đáp ứng các biến cố. Ngược lại, Visual Basic sẽ dừng quá trình xử lý biến cố điều khiển riêng biệt đó.
FillStyle	Nếu định là 0–Single (mặc định), một giá trị chỉ được gán cho một ô được chọn, nếu là 1–Repeat, một giá trị có thể được gán cho nhiều ô được chọn.
FixedCols	Lưu số cột cố định. Giá trị FixedCols tối thiểu là 2 và nhỏ hơn giá trị Cols.
FixedRows	Lưu số hàng cố định. Giá trị FixedRows tối thiểu là 2 và nhỏ hơn giá trị Rows.
FontBold	Giá trị True (mặc định) nếu các giá trị khung lưới hiển thị ở dạng chữ đậm; ngược lại là False.
FontItalic	Giá trị True (mặc định) nếu các giá trị khung lưới hiển thị ở dạng chữ nghiêng; ngược lại là False.
FontName	Tên kiểu phông chữ cho các giá trị khung lưới. Tiêu biểu, bạn sẽ dùng tên phông chữ TrueType trong Windows.
FontSize	Kích cỡ phông chữ tính theo point được dùng cho các giá trị khung lưới.
FontStrikethru	Giá trị True (mặc định) nếu các giá trị khung lưới sẽ hiển thị ở dạng ký tự gạch ngang (các ký tự có đường gạch băng ngang qua); ngược lại là False.
FontUnderline	Giá trị True (mặc định) nếu các giá trị khung lưới hiển thị ở dạng các ký tự gạch dưới; ngược lại là False.
ForeColor	Xác định màu chữ trong các giá trị khung lưới.

GridLines	Giá trị True (mặc định) nếu khung lưới có các đường chia hàng và cột; ngược lại là False.
Height	Chiều cao của khung lưới theo đơn vị twip.
HelpContextID	Nếu bổ sung trợ giúp cảm ngữ cảnh cao cấp tới trình ứng dụng của bạn, thuộc tính HelpContextID sẽ cung cấp giá trị số xác định văn bản trợ giúp.
Highlight	Giá trị True (mặc định) hay False để xác định liệu các ô được chọn sẽ hiện sáng.
Index	Nếu khung lưới là thành phần của mảng điều khiển, thuộc tính Index sẽ cung cấp giá trị chỉ số cho từng điều khiển khung lưới riêng biệt.
Left	Khoảng cách twip từ biên trái cửa sổ Form tới biên trái của khung lưới.
Name	Tên điều khiển. Mặc định, Visual Basic sẽ phát sinh các tên Grid1, Grid2, và ... khi bạn bổ sung lần lượt các khung lưới vào mẫu biểu.
Rows	Số hàng trong khung lưới.
ScrollBars	Giá trị 0–None, 1–Horizontal, 2–Vertical, hay 3–Both (mặc định) để mô tả các thanh cuốn sẽ xuất hiện trong khung lưới nếu khung lưới đòi hỏi số hàng và cột nhiều hơn kích cỡ khung lưới cho phép.
TabIndex	Xác định thứ tự tab tiêu điểm bắt đầu từ 0 và tăng 1 mỗi lần bổ sung điều khiển mới. Bạn có thể thay đổi thứ tự tiêu điểm bằng cách thay đổi giá trị thuộc tính TabIndex của điều khiển. Không có hai điều khiển nào trên cùng mẫu biểu có thể có cùng giá trị TabIndex.
TabStop	Nếu là True, sẽ xác định liệu người dùng có thể nhấn Tab để di chuyển tiêu điểm tới khung lưới này. Nếu là False, khung lưới không thể nhận tiêu điểm.
Tag	Không được Visual Basic sử dụng. Đây là tiện ích dành cho lập trình viên dùng để nhận biết chú thích áp dụng cho khung lưới.
Top	Khoảng cách twip từ cạnh trên cùng khung lưới tới cạnh trên cùng mẫu biểu.
Visible	Giá trị True hay False, cho biết liệu người dùng có thể xem và sử dụng khung lưới.
Width	Độ rộng điều khiển khung lưới.

Mách nước

Điều khiển khung lưới (Grid) cho phép người dùng của bạn xem và cuộn các giá trị nhưng không cho phép người dùng nhập các giá trị mới vào khung lưới. Chương trình của bạn có thể cập nhật khung lưới, nhưng người dùng thì không thể.

Ngoài các giá trị thuộc tính bạn có thể định lúc thiết kế chương trình (các thuộc tính này sẽ liệt kê trong Bảng 23.2), còn có những giá trị thuộc tính chương trình của bạn có thể bổ sung hay điều chỉnh trong thời gian thi hành. Bảng 23.3 sẽ liệt kê các thuộc tính có hiệu lực với mã lệnh trong thời gian thi hành.

Bảng 23.3. *Các thuộc tính có hiệu lực trong thời gian thi hành của điều khiển khung lưới*

Thuộc tính	Mô tả
Col	Số cột, bắt đầu từ 0 của ô được chọn hiện hành.
ColAlignment	Giá trị 0 (mặc định) để canh trái các giá trị ô, giá trị 1 để canh phải, và giá trị 2 để canh giữa. Thuộc tính này chỉ áp dụng cho các ô trong những cột không cố định.
ColWidth	Độ rộng twip của một cột.
FixedAlignment	Giá trị 0 (mặc định) để canh trái các giá trị ô, giá trị 1 để canh phải, và giá trị 2 để canh giữa. Thuộc tính này chỉ áp dụng cho các ô trong những cột cố định.
HighLight	Có giá trị True hoặc False để cho biết liệu người dùng đã chọn ô hay khối ô.
Picture	Trong thời gian thi hành, bạn có thể gán thủ tục LoadPicture() để hiển thị hình ảnh đồ họa trong ô được chọn.
Row	Số hàng, đếm từ 0, của ô được chọn hiện hành.
RowHeight	Chiều cao twip của một hàng.
SelEndCol	Cột cuối cùng bên phải khối được chọn.
SelEndRow	Hàng cuối cùng của khối được chọn.
SelStartCol	Cột đầu tiên bên trái khối được chọn.
SelEndCol	Hàng đầu tiên của khối được chọn.
Text	Giá trị nội dung ô.

Hai giá trị thuộc tính thường sử dụng lúc thi hành chương trình là Row và Col. Hai giá trị thuộc tính này sẽ xác định ô nào bạn đang định dạng và ô nào bạn muốn gán văn bản vào. Trước khi gán giá trị vào một ô trong khung lưới, bạn phải định các giá trị Row và Col vào hàng và cột tương ứng với ô cần gán.

Ghi chú

Nhiều giá trị thuộc tính trong Bảng 23.3 chỉ có ý nghĩa khi người dùng thi hành chương trình, bởi vì cho đến lúc đó, không có ô nào được chọn. Bài thực hành cuối chương sẽ minh họa cách sử dụng các thuộc tính này để khởi tạo Grid và đáp ứng khối các ô được chọn của người dùng.

Củng cố

Điều khiển khung lưới (Grid) tỏ ra đắc dụng trong quá trình hiển thị các bảng giá trị dữ liệu cho người dùng. Điều khiển khung lưới là một điều khiển chuyên biệt được tìm thấy trong tập tin GRID.VBX mà bạn phải thêm vào ứng dụng bất kỳ có sử dụng điều khiển khung lưới.

GIÁM SÁT CON TRỎ Mouse

Khái niệm

Chương trình của bạn có thể đáp ứng quá trình di chuyển và nhấp mouse thông qua các thủ tục biến cố. Đối tượng mouse hỗ trợ giá trị thuộc tính và phương thức có thể sử dụng để giám sát mouse. Một thuộc tính quan trọng bạn muốn điều khiển trong suốt quá trình di chuyển mouse của người dùng là MousePointer. Thuộc tính này xác định cách mouse được nhìn thấy khi người dùng di chuyển mouse trên điều khiển.

Nếu cần, bạn có thể giám sát quá trình di chuyển và nhấp mouse của người dùng. Ví dụ, nếu người dùng nhấp mouse trên một nút lệnh, Visual Basic sẽ đảm bảo rằng thủ tục biến cố Click của nút lệnh tự động thi hành mà bạn không phải lo lắng tìm quá trình nhấp mouse.

Một trong những ứng dụng phổ biến mà chương trình làm việc với mouse là thay đổi hình dạng con trỏ mouse. Trong Windows, thuật ngữ *cursor,* về mặt kỹ thuật, chỉ dùng cho con trỏ mouse, cũng như thuật ngữ *caret* dùng cho con trỏ văn bản. Mặc dù thiết kế ban đầu của Microsoft và cẩm nang kỹ thuật cung cấp các tên “chính xác”, đa số người dùng

và lập trình viên thích gọi con trỏ mouse là *mouse cursor* và con trỏ văn bản là *text cursor*. Chương này sẽ tiếp tục dùng thuật ngữ riêng.

Bảng 23.4 trình bày danh sách hình dạng con trỏ mouse có thể hiển thị. Phần lớn điều khiển có thuộc tính MousePointer. Giá trị thuộc tính MousePointer của từng điều khiển, được mô tả trong Bảng 23.4, sẽ xác định hình dạng con trỏ mouse nào sẽ hiển thị khi người dùng di chuyển mouse trên điều khiển đó.

Bảng 23.4. *13 giá trị con trỏ mouse*

Giá trị	Mô tả
0–Default	Con trỏ mouse của điều khiển có hình dạng mặc định. Mỗi điều khiển có hình dạng con trỏ mouse mặc định riêng. Đa số điều khiển sử dụng con trỏ mouse chung cho hình dạng mặc định.
1–Arrow	Con trỏ mouse có dạng mũi tên đặc thù. (Thông thường, nó giống hình dạng con trỏ mouse mặc định của điều khiển.)
2–Cross	Con trỏ có dạng cái kéo.
3–I–Beam	Con trỏ mouse có dạng gạch đứng thường dùng như một con trỏ văn bản.
4–Icon	Con trỏ mouse có dạng 2 hình vuông lồng nhau, hình vuông trong nhỏ có màu đen.
5–Size	Con trỏ mouse có dạng giống dấu + với 4 mũi tên chỉ ra 4 hướng.
6–Size NE SW	Con trỏ mouse có dạng mũi tên chéo đông bắc và tây nam.
7–Size N S	Con trỏ mouse dạng đứng theo hướng bắc nam.
8–NW SE	Con trỏ mouse có dạng mũi tên chéo theo hướng tây bắc và đông nam.
9–W E	Mũi tên dọc chỉ hướng đông và tây.
10–Up Arrow	Mũi tên hướng lên.
11–Hourglass	Hình dạng đồng hồ cát.
12–No Drop	Ký hiệu đi đường quen thuộc vòng tròn "No" với một đường chéo vạch ngang nó.

Phụ thuộc vào trình ứng dụng, bạn có thể giữ lại thao tác nhấp trên điều khiển của người dùng hay không. Ví dụ, bạn muốn bỏ qua tất cả thao tác nhấp một nút lệnh in báo cáo hoạt động hàng ngày cho đến sau 5 giờ chiều, giờ đóng cửa. Trình ứng dụng của bạn có thể đáp ứng quá trình nhấp nút lệnh sau 5 giờ chiều và không đáp ứng nếu trước 5 giờ chiều.

Trong suốt thời gian nút lệnh bị bỏ qua, bạn có thể định hình dạng MousePointer của nút lệnh là con trỏ mouse 12–No Drop. Vì vậy, con trỏ mouse của người dùng sẽ thay đổi thành dạng No Drop bất cứ khi nào người dùng di chuyển con trỏ mouse trên nút lệnh đó. Con trỏ mouse có thể vẫn là con trỏ mũi tên chuẩn với các điều khiển khác. Sau 5 giờ chiều, bên trong thủ tục nhấp, bạn có thể thực hiện quá trình nhấp nút lệnh để in báo cáo và định thuộc tính MousePointer thành 1-Default (hay 1–Arrow, cùng hình dạng con trỏ mouse bởi vì con trỏ mouse mặc định là mũi tên cho các điều khiển nút lệnh).

Bảng 23.5 sẽ liệt kê các tên hằng trong tập tin CONSTANT.TXT mà bạn có thể dùng để định các giá trị thuộc tính MousePointer.

Bảng 23.5. *Tên các giá trị thuộc tính MousePointer trong CONSTANT.TXT*

Thuộc tính	Mô tả
DEFAULT	Dạng con trỏ mouse mặc định của điều khiển
ARROW	Mũi tên con trỏ mouse điển hình
CROSSHAIR	Một cái kéo
IBEAM	Con trỏ văn bản điển hình
ICON_POINTER	Biểu tượng hình vuông trong 1 hình vuông
SIZE_POINTER	Dấu + với các mũi tên hướng ra 4 hướng
SIZE_NE_SW	Mũi tên chéo chỉ hướng đông bắc đến tây nam
SIZE_N_S	Mũi tên đứng chỉ hướng bắc nam
SIZE_NW_SE	Mũi tên chéo chỉ hướng tây bắc đến đông nam
SIZE_W_E	Mũi tên chỉ hướng đông tây
UP_ARROW	Mũi tên hướng lên
HOURGLASS	Đồng hồ cát
NO_DROP	Vòng tròn "No" với một đường gạch ngang

Tóm tắt

Ví dụ 23.2 trình bày mã lệnh trong trình ứng dụng MOUSECH.VBP. Chương trình minh họa cách con trỏ mouse có thể thay đổi phụ thuộc vào giá trị của nút tùy chọn. Chương trình đưa ra các kết quả trước và sau 5 giờ chiều như được mô tả ở phần trước.

Củng cố

Đa số điều khiển hỗ trợ thuộc tính MousePointer xác định hình dạng con trỏ mouse khi người dùng di chuyển mouse trên điều khiển.

Ví dụ 23.2. *Mã lệnh thay đổi hình dạng mouse phụ thuộc vào giá trị các nút tùy chọn.*

```
Sub optAfter_Click ()
' After 5:00 pm so fix mouse over command
cmdReport.MousePointer = ARROW
End Sub

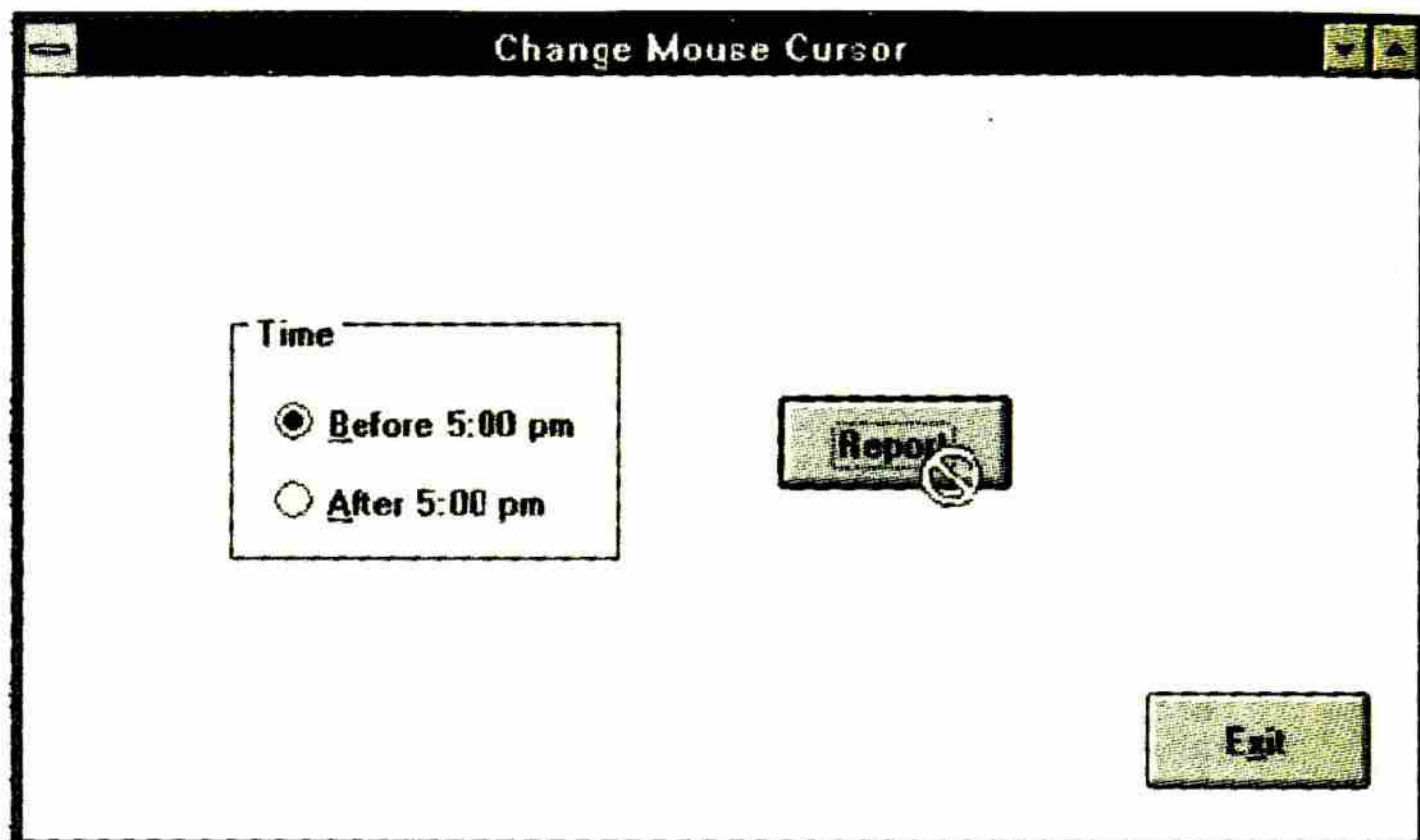
Sub optBefore_Click ()
' Before 5:00 pm so negate mouse over command
cmdReport.MousePointer = NO_DROP
lblReport.Visible = False
End Sub

Sub cmdExit_Click ()
End
End Sub

Sub cmdReport_Click ()
' Beep if it's before 5:00
If optBefore.Value = True Then
Beep
lblReport.Visible = False
Else
lblReport.Visible = True
End If
End Sub
```

Kết quả

Hình 23.4 mô tả những gì sẽ xảy ra với hình dạng con trỏ mouse khi người dùng di chuyển con trỏ mouse trên nút lệnh trước 5 giờ chiều.



Hình 23.4. *Con trỏ mouse báo cho người dùng biết nút lệnh sẽ không đáp ứng.*

Phân tích

Khi chương trình bắt đầu, nút tùy chọn đầu tiên tên optBefore sẽ định là True. Vì vậy, trình ứng dụng giả sử thời gian lúc đó trước 5 giờ chiều và giá trị MousePointer của nút lệnh được định là NO_DROP như minh họa trong Hình 23.4.

Ghi chú

Chương trình có thể sử dụng hàm Time$() kiểm tra thời gian thực tế để xem liệu nút lệnh có thể hoạt động hay không. Tuy nhiên, các nút tùy chọn cho bạn phương thức thực tập dễ dàng hơn với 2 con trỏ mouse; bạn không phải đợi cho đến 5 giờ chiều để xem sự khác nhau.

Ngay khi người dùng nhấp tùy chọn After 5:00 p.m. tên optAfter, thủ tục biến cố optAfter_Click() sẽ thay đổi thuộc tính MousePointer của nút lệnh thành ARROW trong dòng 3.

Khi người dùng nhấp nút lệnh Report, thủ tục biến cố cmdReport_Click() của nút lệnh sẽ thi hành (dòng 16). Nếu nút tùy chọn Before 5:00 pm được thiết đặt, dòng 19 sẽ phát tiếng bíp. Nếu nút tùy chọn After 5:00 pm được ấn định, một nhãn nhỏ sẽ xuất hiện ở dưới nút lệnh báo cho bạn báo cáo có thể in được. (Nhãn tên lblReport, được ấn định là False khi chương trình mới bắt đầu.)

CHẶN QUÁ TRÌNH NHẤP VÀ DI CHUYỂN Mouse

Khái niệm

Mouse hỗ trợ 3 biến cố trả về thông tin nhấp và di chuyển để người dùng liên lạc với các thủ tục biến cố qua mẫu biểu. Không giống nhiều thủ tục biến cố, các thủ tục biến cố mouse sử dụng những đối số trả lại thông tin như vị trí mouse và nút được nhấp.

Bảng 23.6 liệt kê các biến cố thường được hỗ trợ nhất bởi các trình ứng dụng liên quan đến mouse. Những biến cố này trả lại biến cố mà từ đó bạn có thể viết thủ tục biến cố để đáp ứng các lệnh mouse của người dùng.

Bảng 23.6. *Các biến cố thường được kết hợp nhiều nhất với mouse*

Biến cố	Mô tả
DblClick	Phát sinh khi người dùng nhấp đúp một nút mouse.
MouseDown	Phát sinh khi người dùng nhấn một nút mouse.
MouseMove	Phát sinh khi người dùng di chuyển mouse.
MouseUp	Phát sinh khi người dùng thả một nút mouse.

Các thủ tục biến cố MouseDown, MouseMove, và MouseUp mở danh sách các đối số khi bạn chọn chúng trong cửa sổ Code. Giả sử bạn muốn đáp ứng quá trình nhấp mouse của người dùng khi người dùng nhấp mouse trên mẫu biểu frmApp. Sau đây là mã lệnh đầu cuối của hai thủ tục biến cố mà Visual Basic sẽ mở cho bạn khi bạn chọn thủ tục MouseDown, MouseMove, hay MouseUp từ hộp danh sách Proc xổ xuống của cửa sổ Code:

```
Sub frmApp_MouseDown
(Button As Integer, Shift As Integer, X As Single, Y As Single)
End Sub
Sub frmApp_MouseMove
(Button As Integer, Shift As Integer, X As Single, Y As Single)
End Sub
Sub frmApp_MouseUp
(Button As Integer, Shift As Integer, X As Single, Y As Single)
End Sub
```

Danh sách đối số của các thủ tục này quá dài khiến bạn phải cuộn ngang cửa sổ Code để xem toàn bộ danh sách đối số.

Bảng 23.7 sẽ liệt kê nội dung mô tả cho 4 giá trị đối số này. Những giá trị được trả lại trong các đối số sẽ cho biết thao tác nhấn hay thả mouse của người dùng. Mã lệnh của bạn có thể kiểm tra giá trị đối số để xác định những gì cần biết. Ví dụ, nếu bạn muốn biết vị trí mouse khi người dùng nhấp mouse trên mẫu biểu, các đối số X, Y sẽ mô tả tọa độ twip của góc trái trên mẫu biểu mà người dùng đã nhấp.

Bảng 23.7. *Các đối số MouseDown() và MouseUp()*

Đối số	Mô tả
Button	Lưu số cho biết nút mouse được nhấn. Giá trị 1 cho biết nút trái được nhấn, 2 cho biết nút phải được nhấn, và 4 cho biết cả hai nút trái phải hay nút giữa với mouse 3 nút được nhấn.
Shift	Mô tả phím Shift được nhấn cùng lúc người dùng nhấp hay di chuyển nút mouse. Thuộc tính Shift có giá trị 1 nếu người dùng nhấn phím Shift trong khi nhấp mouse, 2 nếu giữ phím Ctrl, và 4 nếu giữ phím Alt. Thuộc tính Shift có thể có giá trị khác 1, 2, hay 4, và bằng tổng 2 hay nhiều số tương ứng với các phím nhấn mô tả ở trên. Ví dụ, nếu người dùng nhấn phím Alt cùng lúc nhấn nút mouse, giá trị thuộc tính Shift sẽ là 5.
X	Cho biết tọa độ ngang của con trỏ mouse tính theo đơn vị twip khi người dùng nhấp hay di chuyển nút mouse.
Y	Cho biết tọa độ dọc của con trỏ mouse tính theo đơn vị twip khi người dùng nhấp hay di chuyển nút mouse.

Nếu người dùng nhấp đúp mouse, Visual Basic sẽ phát sinh cả hai biến cố MouseUp và DblClick. Thủ tục biến cố MouseUp trả lại vị trí và thông tin nút nhấp đúp (xem Bảng 23.6), và biến cố DblClick chứa mã lệnh bạn muốn thi hành vào lúc nhấp đúp mouse.

Biến cố MouseUp xảy ra mỗi lần người dùng thả nút mouse, không cần biết người dùng thả nút nhấn mouse do quá trình nhấp đơn hay nhấp đúp. Vì vậy, nếu muốn đáp ứng biến cố MouseUp của quá trình nhấp đơn mà không phải biến cố MouseUp của quá trình nhấp đúp, bạn phải đặt một biến module bên trong thủ tục biến cố DblClick để kiểm tra xem loại nhấp đơn hay nhấp đúp đã phát sinh biến cố MouseUp.

Ghi chú

Phụ thuộc vào máy tính của bạn, Visual Basic có thể phát sinh thủ tục biến cố di chuyển mouse mỗi lần người dùng di chuyển mouse đi một khoảng 10 đến 15 twip trên mẫu biểu. Visual Basic sẽ không phát sinh biến cố di chuyển mouse cho quá trình di chuyển từng twip; đa số máy tính không thể duy trì các biến cố thi hành nhanh như thế.

Tóm tắt

Ví dụ 23.3 trình bày thủ tục biến cố MouseUp() mẫu minh họa cách mã lệnh có thể sử dụng các đối số thủ tục để biết thêm về quá trình di chuyển mouse. Mã lệnh in thông tin mouse. Bạn có thể viết bộ thủ tục biến cố mouse tương tự cho các biến cố mouse khác, ghi lại những hành động mouse khi bạn di chuyển, nhấp và nhấp đúp mouse.

Củng cố

Các thủ tục biến cố nhấp và di chuyển mouse cần thông tin bổ sung khi biến cố mouse xảy ra. Thông qua danh sách đối số trong thủ tục biến cố mouse, bạn có thể xác định nút nào người dùng đã nhấn, phím nào người dùng đang nhấn tại thời điểm nhấp hay di chuyển mouse và biết chính xác vị trí mouse trên mẫu biểu khi người dùng phát sinh biến cố.

Ví dụ 23.3. *Mã lệnh kiểm tra mouse sau khi thả nút nhấn.*

```
1: Sub frmApp_MouseUp
(Button As Integer, Shift As Integer, X As Single, Y As Single)
2: ' Display text on the printer that describes the
3: ' mouse button press. The semicolon at the end of
4: ' the Printer.Print statements forces Visual
5: ' to keep the printer cursor on the same Print
6: ' line.
7:
8: ' Tell the user about the button pressed
9: Printer.Print "Up: The button you released was the ";
10: Select Case Button
11: Case 1: Printer.Print "Left ";
12: Case 2: Printer.Print "Right ";
13: Case 4: Printer.Print "Middle ";
14: End Select
15: Printer.Print "button"
16:
17: Printer.Print "The mouse was at X position: ";
18: Printer.Print X;
19: Printer.Print "and Y position: ";
20: Printer.Print Y
21: ' Print a blank line to separate for subsequent output
22: Printer.Print
23:
24: End Sub
```

Phân tích

Dòng 1 tập hợp các đối số thả nút mouse. Đối số báo cho thủ tục biến cố rằng người dùng vừa thả nút hoặc một phím được nhấn, và tọa độ mouse tính theo twip chính xác khi người dùng thả mouse.

Phần còn lại, thủ tục biến cố xử lý để in thông tin mouse dựa trên các đối số. Hãy nhớ rằng thủ tục biến cố này sẽ thi hành mỗi lần người dùng thả nút mouse, cho nên kết quả xuất ra máy in sẽ tiếp tục khi người dùng tiếp tục nhấp mouse.

Bài tập

Kiến thức tổng quát

1. Có bao nhiêu điều khiển thanh cuốn khác nhau?
2. Thanh cuốn loại bỏ quá trình nhập nội dung xác định như thế nào?
3. Các thuộc tính LargeChange và SmallChange nói lên điều gì?
4. Người dùng kích hoạt thuộc tính LargeChange như thế nào để thay đổi giá trị của thanh cuốn?
5. Các thuộc tính Min và Max cho biết điều gì?
6. Những ký tự đầu tiên trong tên của các thanh cuốn là gì?
7. Đúng hay Sai: Bạn có thể khởi tạo thanh cuốn bằng cách dùng một biến được định nghĩa với kiểu dữ liệu nguyên.
8. Bạn phải làm gì để thêm điều khiển khung lưới (Grid) vào cửa sổ Toolbox?
9. Tên điều khiển tập tin chuyên biệt giữ điều khiển khung lưới là gì?
10. Đúng hay Sai: Người dùng có thể đọc, thay đổi, và nhập các giá trị mới vào trong khung lưới.
11. Ô là gì?
12. Đúng hay Sai: Khi bạn điều chỉnh kích cỡ vật lý của khung lưới trên mẫu biểu, kích cỡ khung lưới phải đủ lớn để hiển thị tất cả các hàng và cột của khung lưới.
13. Khi nào các thanh cuốn có thể xuất hiện trên khung lưới?
14. Hai giá trị thuộc tính nào chỉ có tác dụng lúc chạy chương trình sẽ xác định một ô riêng biệt muốn thay đổi hay khởi tạo giá trị bằng mã lệnh?
15. Bốn giá trị thuộc tính định nghĩa các ô được chọn là gì?
16. Có bao nhiêu hình dạng mouse khác nhau có thể xuất hiện trên một điều khiển?
17. Thuộc tính nào mô tả hình dạng con trỏ mouse?
18. Về mặt kỹ thuật, con trỏ văn bản được gọi là gì?

19. Về mặt kỹ thuật, con trỏ mouse được gọi là gì?
20. Khi nào các giá trị thuộc tính MousePointer là 0–Default và 1–Arrow khác nhau?
21. Con trỏ mouse nào có ích cho quá trình nhắc người dùng đợi một vài giây cho đến khi quá trình tính toán hoàn thành?
22. Các biến cố di chuyển mouse sẽ phát sinh thông tin phụ mô tả các nút và vị trí mouse?
23. Đúng hay Sai: Khi người dùng nhấp đúp một nút mouse, thực tế hai biến cố sẽ thực hiện.

Tìm lỗi kỹ thuật

24. An Huy muốn dùng điều khiển khung lưới để lưu trọng lượng hóa chất trong một thí nghiệm hóa học dưới dạng một bảng 5 hàng, 5 cột. Sau khi An Huy khởi tạo bảng thông qua mã lệnh, và nhấn một nút lệnh, anh ấy muốn mã lệnh điền nội dung bảng vào các hàng và cột để người dùng không thể chọn những ô bổ sung. Đáng buồn là Visual Basic đưa ra một giới hạn không cho phép An Huy tự do điền vào tất cả hàng và cột. Số hàng và số cột tối đa có thể điền trong bảng là bao nhiêu?

Lập trình...

25. Giả sử bạn đang viết một chương trình cần người dùng nhập vào giá trị nhiệt độ trong khoảng từ 32 đến 212 độ. Bạn muốn người dùng nhập vào nhiệt độ bằng cách đọc vị trí tương đối trên thanh cuốn. Khi người dùng nhấp một trong các mũi tên trên thanh cuốn, thanh cuốn sẽ được điều chỉnh 3 độ. Khi người dùng nhấp trong vùng thân thanh cuốn, thanh cuốn cần phải điều chỉnh 8 độ. Các thuộc tính Min, Max, SmallChange, và LargeChange thích hợp là gì?

Phần nâng cao

Hãy viết chương trình có một nút lệnh với nhãn là Change Mouse để hiển thị các con trỏ mouse khác nhau mỗi lần người dùng nhấp nút. Hãy dùng một biến module để lưu giá trị của con trỏ mouse hiện hành.

Bài 24

Dò tìm và xử lý lỗi

- **Quá trình dò lỗi**
- **Chương trình gỡ rối (debugger)**
- **Chuyển sang chế độ Break**
- **Sử dụng cửa sổ Immediate**

Cuốn sách hoàn tất bằng cách hướng dẫn cho bạn cách tạm dừng quá trình thi hành mã lệnh và những điều khiển thông thường. Khi viết càng nhiều chương trình, kỹ năng Visual Basic của bạn sẽ cải thiện hơn. Vấn đề là lỗi lập trình cũng sẽ gia tăng. Hầu hết các lập trình viên đều gặp lỗi trong quá trình thi hành mã lệnh. Những lỗi này được hiểu như lỗi kỹ thuật, nhưng đôi khi lại rất khó tìm.

Visual Basic bao gồm nhiều công cụ dò lỗi giúp bạn dò tìm, phát hiện và sửa đổi lỗi kỹ thuật. Quá trình dò lỗi kỹ thuật trong Visual Basic thật tuyệt vời. Có lẽ nói hơi quá, nhưng các công cụ dò lỗi mà Visual Basic cung cấp chắc chắn sẽ giúp quá trình tìm lỗi kỹ thuật dễ dàng hơn so với những ngôn ngữ lập trình trước đây của Visual Basic.

QUÁ TRÌNH DÒ LỖI

Khái niệm

Nhiều loại lỗi có thể diễn ra. Một số lỗi dễ dàng tìm thấy, nhưng một số thì không dễ gì. Các công cụ dò lỗi của Visual Basic sẽ giúp bạn đột nhập vào mã lệnh chương trình để dò nhiều loại lỗi khó tìm.

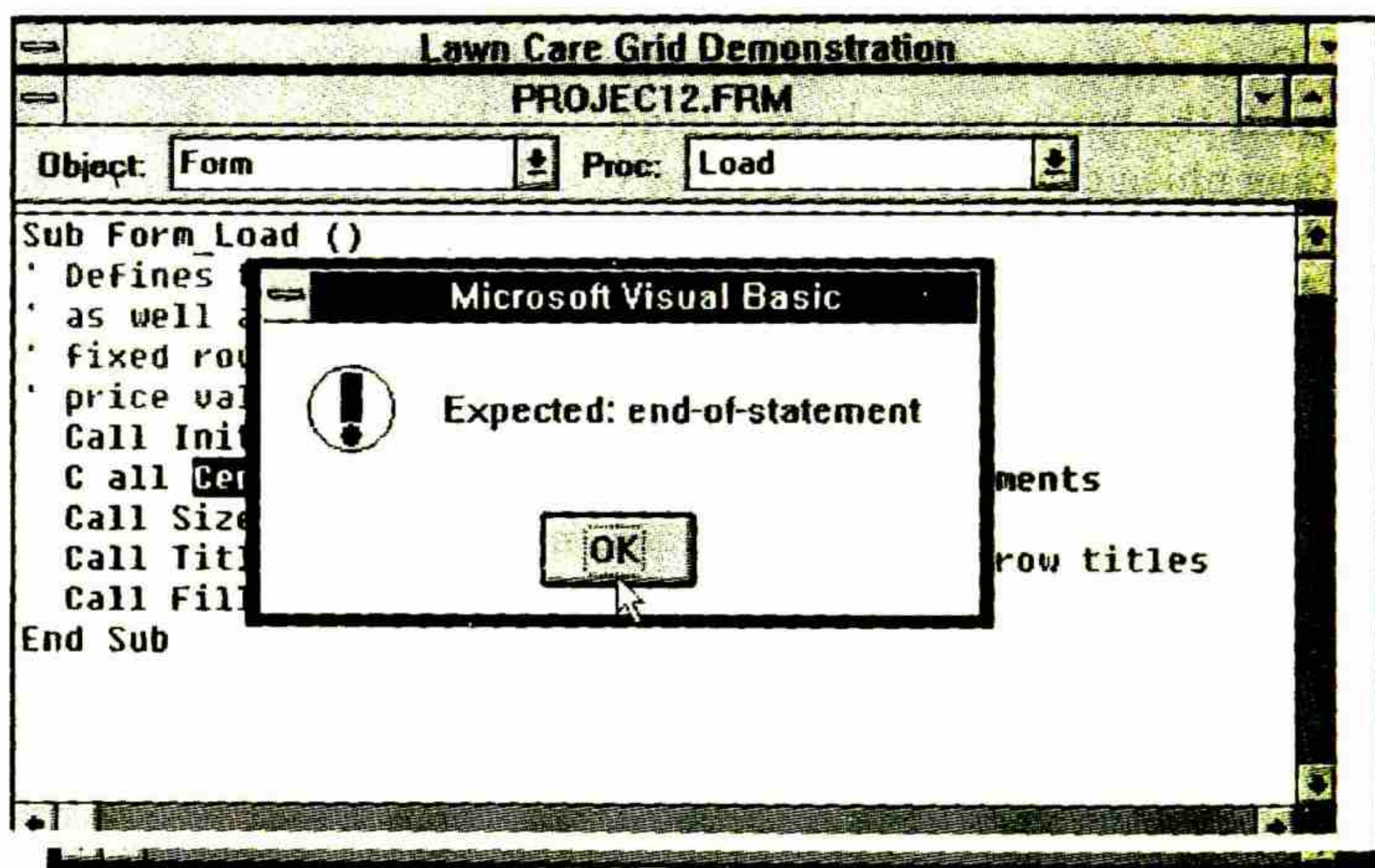
Khái niệm mới

Chương trình gỡ rối là công cụ tích hợp giúp bạn tìm các lỗi kỹ thuật trong chương trình.

Như đã đề cập trong Bài 1, có hai nhóm lỗi phổ biến bạn có thể gặp khi dùng Visual Basic. Chương trình gỡ rối giúp bạn tìm và hiệu chỉnh lỗi. Bạn sẽ tìm thấy cả lỗi cú pháp lẫn lỗi logic.

Lỗi cú pháp là loại lỗi dễ tìm nhất. Lỗi cú pháp thường hay xảy ra bởi lẽ đánh vần sai tên lệnh, hàm hoặc phương thức. Lỗi cú pháp cũng có thể xuất hiện vì bạn sắp xếp lại ngữ pháp của vòng lặp như vòng lặp For với tùy chọn Step đặt trước từ khóa To của vòng lặp.

Khi lập trình, Visual Basic sẽ phát hiện lỗi cú pháp cho bạn. Nếu bạn định tùy chọn Options Environment Syntax Checking là Yes, Visual Basic thực sự kiểm tra lỗi cú pháp mỗi lần nhấn phím Enter ở cuối dòng mã lệnh có lỗi. Hình 24.1 sẽ trình bày cách Visual Basic hiển thị thông báo lỗi cho người dùng.



Hình 24.1. *Lập trình viên vừa nhập lỗi cú pháp cần thực hiện đúng vấn đề.*

Khi bạn gặp hộp thông báo lỗi cú pháp, nhấn phím Enter để thoát khỏi hộp thông báo và thực hiện đúng vấn đề. Thông thường, bạn sẽ thấy ngay từ sai và sửa đúng lỗi. Nếu bạn không thể tìm thấy lỗi, hoặc giả bạn muốn trở lại vấn đề sau đó, Visual Basic sẽ cho phép bạn tiếp tục viết phần còn lại của chương trình vốn không hiển thị lại thông báo lỗi trừ phi bạn thay đổi trên dòng lệnh lỗi đó và tiếp tục sai sau khi hiệu chỉnh lỗi.

Lưu ý

Qua Hình 24.1 ta thấy, Visual Basic không gọi lỗi là lỗi cú pháp nhưng từ thông báo lỗi bạn sẽ biết rằng cú pháp của câu lệnh (ngữ pháp hoặc cách đánh vần) là sai.

Nếu bạn ấn định tùy chọn Options Environment Syntax Checking là No, Visual Basic sẽ bỏ qua tất cả lỗi cú pháp đến khi bạn chạy chương trình. Trong thời gian chạy chương trình, Visual Basic sẽ hiển thị hộp thông báo lỗi cú pháp khi Visual Basic gặp chúng.

Loại lỗi thứ hai bạn sẽ gặp là lỗi logic. Lỗi logic thì khó tìm hơn. Lỗi logic là lỗi về hành động.

Lỗi logic xuất hiện khi chương trình dường như hiểu mọi thứ bạn đã nhập nhưng đưa ra kết quả sai. Chẳng hạn, bạn đã viết chương trình in séc thanh toán và chương trình thực hiện lỗi tính toán là tăng gấp đôi số tiền thưởng cho các nhân viên làm việc cho công ty trên 10 năm. Nếu bạn chạy chương trình và in séc, có lẽ bạn sẽ không nhận ra những lỗi này. Tuy nhiên, bạn sẽ nhận ra lỗi khi nhân viên thắc mắc.

Đôi khi, lỗi logic sẽ không xuất hiện đến khi chúng ta bắt đầu sử dụng chương trình. Bạn sẽ gặp ngay lỗi cú pháp khi chạy chương trình vì Visual Basic giúp bạn tìm chúng thông qua hộp thông báo. Chúng ta phải tìm và tập hợp lỗi logic. Khi bạn vận hành chương trình có vấn đề trong tính toán, trong vòng lặp hoặc thủ tục, bạn phải quay lại cửa sổ Code và tìm ra vấn đề. Trong những chương trình lớn, lỗi logic sẽ rất khó tìm.

Lỗi logic có thể xảy ra bởi vì bạn quên khởi tạo giá trị đúng cho biến, bạn đã không lặp đủ các giá trị trong điều khiển bảng hoặc danh sách, hoặc đã đảo kiểu dữ liệu khi định nghĩa biến. Mặc dù lỗi logic thường khó tìm, song trình gỡ rối được tích hợp trong Visual Basic, trình bày trong phần kế tiếp, sẽ giúp bạn tìm các lỗi đó.

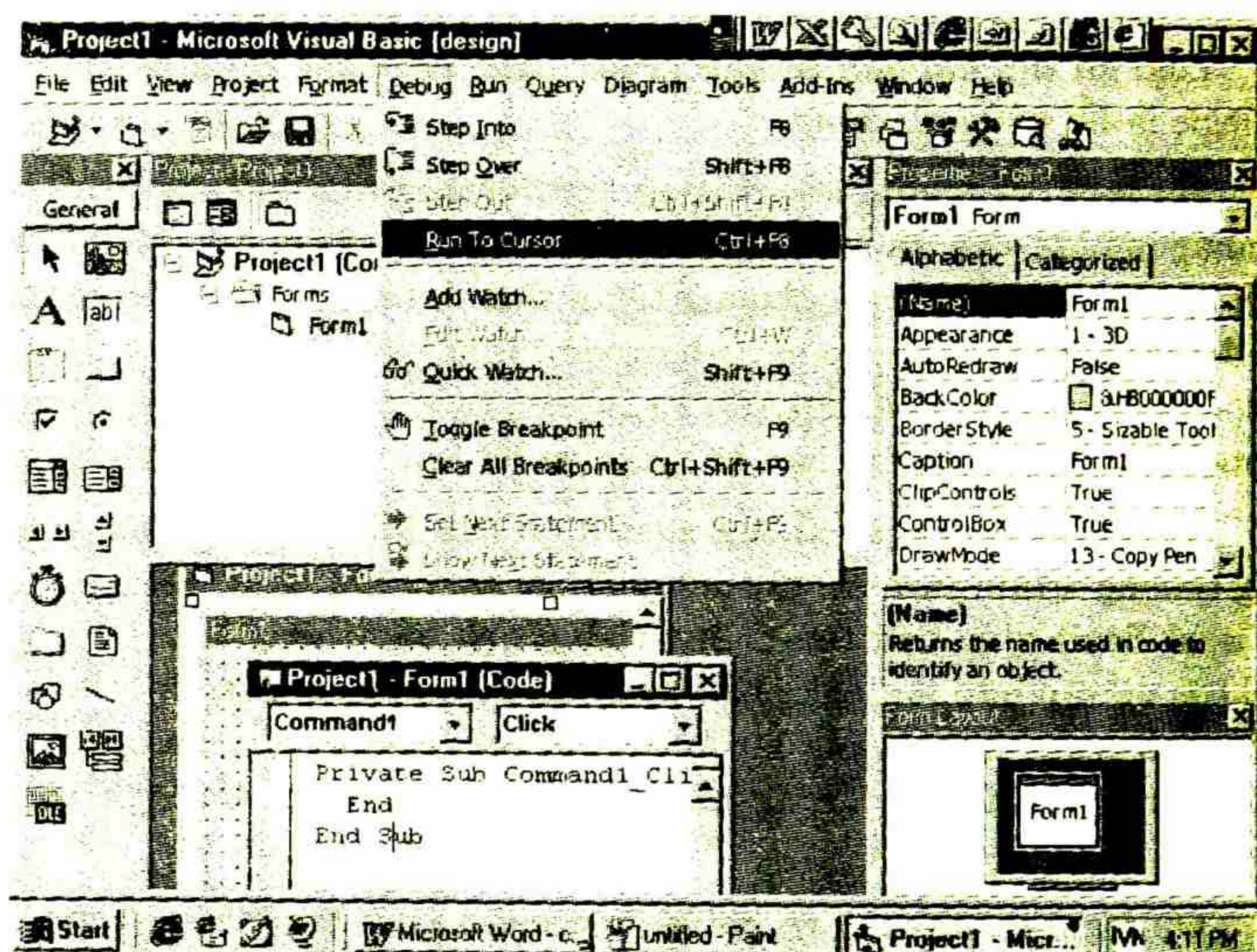
Củng cố

Có hai nhóm lỗi: lỗi cú pháp và lỗi logic. Visual Basic sẽ tìm tất cả lỗi cú pháp trong chương trình. Lỗi logic là phần việc của bạn. Trình gỡ rối (debugger) sẽ giúp bạn tìm lỗi logic.

CHƯƠNG TRÌNH GỠ RỐI

Khái niệm

Menu Debug đưa ra danh sách tất cả các lệnh dò lỗi của Visual Basic. Dùng lệnh dò lỗi, bạn có thể dừng chương trình trước khi thi hành từng dòng lệnh, có thể thấy giá trị biến và điều khiển, thực hiện mã lệnh từng bước bắt đầu từ dòng lệnh bất kỳ.



Hình 24.2. *Các công cụ dò lỗi có thể dùng trên menu Debug của thanh menu.*

Hình 24.2 minh họa menu Debug. Các công cụ dò lỗi trong Visual Basic rất hoàn hảo và cho phép bạn thi hành chương trình bằng cách sử dụng một trong nhiều phương thức hỗ trợ quá trình phân tích từng đoạn mã lệnh khác nhau.

Khái niệm mới

Visual Basic chuyển đến chế độ dừng Break khi bạn ngừng chương trình đang thi hành.

Các tùy chọn menu này đều có hiệu lực trong chế độ Break. Sau đây là ba chế độ trong chương trình Visual Basic:

- Chế độ *Design* (thiết kế)
- Chế độ *Runtime* (lúc thi hành)
- Chế độ *Break* (dừng)

Visual Basic sẽ báo cho bạn biết chế độ nào là chế độ hiện hành bằng cách hiển thị từ *design, run,* hoặc *break* trong thanh tiêu đề của Visual Basic phía trên màn hình Visual Basic. Khi bạn thiết kế chương trình, chương trình sẽ ở trong chế độ Design; khi bạn hoặc người dùng chạy chương trình, chương trình sẽ ở chế độ Runtime, và khi bạn dừng chương trình dùng trình gỡ rối để phân tích các vấn đề, chương trình sẽ chuyển sang chế độ Break.

Bài này chỉ đề cập đến chế độ Break. Trong chế độ Break, chương trình vẫn duy trì giá trị biến và điều khiển. Vì vậy, bạn có thể ngừng bất cứ lúc nào để xem giá trị dữ liệu từ dòng lệnh bất kỳ. Bằng cách so sánh các giá trị với những giá trị bạn mong đợi, bạn có thể biết được vấn đề xảy ra ở đâu.

Củng cố

Trong ba chế độ Visual Basic, bạn sẽ dùng chế độ Break để dò lỗi chương trình. Khi chuyển sang chế độ Break, chương trình sẽ dừng tại bất kỳ dòng lệnh nào trong quá trình thi hành của chương trình song vẫn giữ lại toàn bộ giá trị biến và giá trị điều khiển được gán đến thời điểm đó.

CHUYỂN SANG CHẾ ĐỘ BREAK

Khái niệm

Có nhiều cách để chuyển sang chế độ Break. Cách phổ biến nhất là định một điểm dừng (hay còn gọi là điểm ngắt). Có thể dừng một chương trình trong lúc thi hành bằng cách ngắt quá trình thi hành chương trình bằng nhấn tổ hợp phím Ctrl+Break hay chọn lệnh Break từ menu Run lúc chạy chương trình.

Bạn luôn chuyển sang chế độ Break từ chế độ Runtime. Chỉ sau khi bắt đầu quá trình thi hành chương trình, chế độ Break mới có hiệu lực, bởi vì chỉ trong thời gian thi hành, biến và điều khiển mới được khởi tạo. Sau đây là một số cách bạn có thể chuyển từ chế độ Runtime sang chế độ Break:

- Nhấn Ctrl+Break trong lúc thi hành chương trình tại nơi bạn muốn chuyển sang chế độ Break. Quá trình dừng tại một dòng mã lệnh riêng biệt thực hiện khi sử dụng Ctrl+Break.
- Chọn lệnh Run Break trên thanh menu.
- Nhấp nút Break trên thanh công cụ (nút có biểu tượng bàn tay).
- Trong chế độ Design (thiết kế) hay chế độ Break (dừng), hãy đặt một điểm dừng (điểm ngắt) trên dòng lệnh bạn muốn quá trình thi hành sẽ dừng. Bằng cách đặt điểm dừng, bạn có thể xác định chính xác dòng lệnh mà Visual Basic sẽ chuyển sang chế độ dừng.
- Hộp thoại Add Watch của menu Debug cho phép bạn xác định một biểu thức dừng mà Visual Basic sẽ kiểm tra và dùng để dừng quá trình thi hành chương trình khi biểu thức trả về giá trị True.
- Nếu một lỗi xảy ra trong thời gian thi hành chương trình, như là phép chia cho 0 mà không có định nghĩa toán tử tương ứng, Visual Basic sẽ chuyển sang chế độ Break ngay tại dòng lệnh tạo ra lỗi. Đôi khi, loại lỗi này được gọi là lỗi *runtime* và thuộc nhóm lỗi logic.

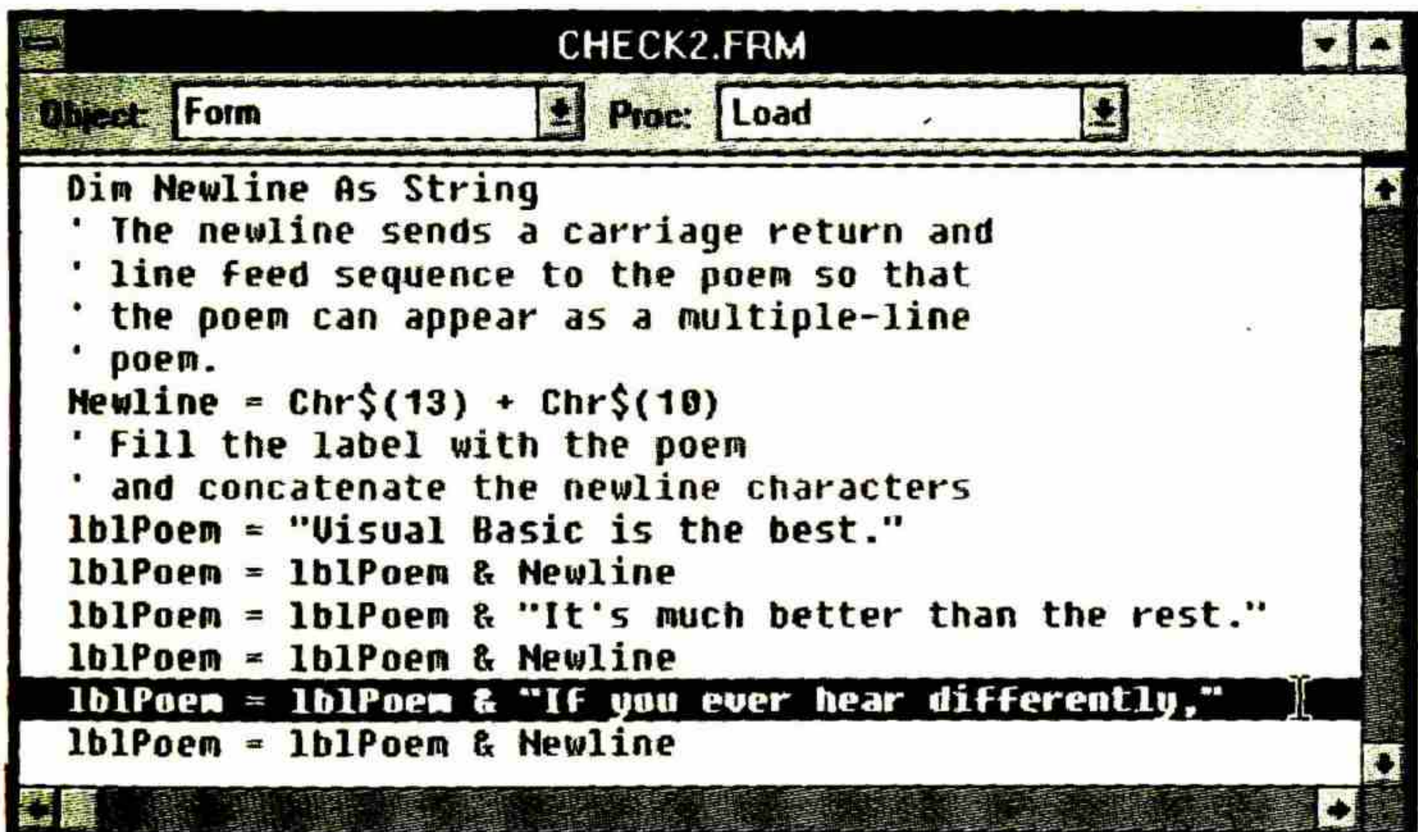
Cách phổ biến và chính xác nhất để chuyển sang chế độ Break là đặt một điểm dừng. Để đặt một điểm dừng, hãy tìm dòng lệnh bạn muốn dừng quá trình thi hành, và đặt một điểm dừng tại dòng lệnh riêng đó. Hãy thực hiện các bước sau để đặt một điểm dừng:

1. Nạp project CHECK2.VBP.
2. Nhấn F7 mở cửa sổ Code. Visual Basic sẽ hiển thị mã lệnh trong thủ tục biến cố Form_Load().
3. Hãy tìm dòng lệnh sau trong Form_Load():

```
lblPoem = lblPoem & "If you ever hear differently,".
```

4. Di chuyển con trỏ mouse đến dòng lệnh và nhấp nút mouse. Con trỏ văn bản sẽ xuất hiện ở vị trí nhấp mouse.

5. Chọn lệnh Debug Toggle Breakpoint để đặt điểm dừng. Từ lệnh menu bạn sẽ thấy F9 là phím truy xuất nhanh của lệnh này. Tương tự, quá trình nhấp biểu tượng bàn tay trên thanh công cụ cũng sẽ đặt một điểm dừng trên dòng lệnh này. Hình 24.3 trình bày cách cửa sổ mã lệnh của bạn sẽ xuất hiện. Visual Basic sẽ đổi màu dòng lệnh để bạn biết một điểm dừng sẽ được đặt trên dòng lệnh trong quá trình thi hành chương trình.



Hình 24.3. *Visual Basic hiện sáng tất cả điểm dừng.*

Bạn có thể bỏ điểm dừng bằng cách chọn lệnh Debug Toggle Breakpoint (hay nhấn F9) lại một lần nữa. Bạn đặt nhiều điểm dừng khi cần trong một chương trình. Bây giờ hãy gỡ bỏ điểm dừng. Bằng cách đặt điểm dừng, bạn đang đòi hỏi Visual Basic dừng chương trình và chuyển sang chế độ dừng khi quá trình thi hành chương trình gặp dòng lệnh đó. Hãy đóng cửa sổ Code và chạy chương trình bằng cách nhấn phím F5.

Ngay khi bạn nhấn F5 để chạy chương trình, Visual Basic sẽ thi hành chương trình. Ít ra, Visual Basic cũng sẽ thi hành mã lệnh cho đến khi gặp điểm dừng đầu tiên. Ngay khi gặp dòng lệnh có điểm dừng, Visual Basic sẽ chuyển sang chế độ Break trước khi dòng lệnh có điểm dừng được thi hành. Bạn có thể xác định chế độ Break bằng cách đọc thanh tiêu đề của Visual Basic và thấy từ **(break)** ở đầu màn hình.

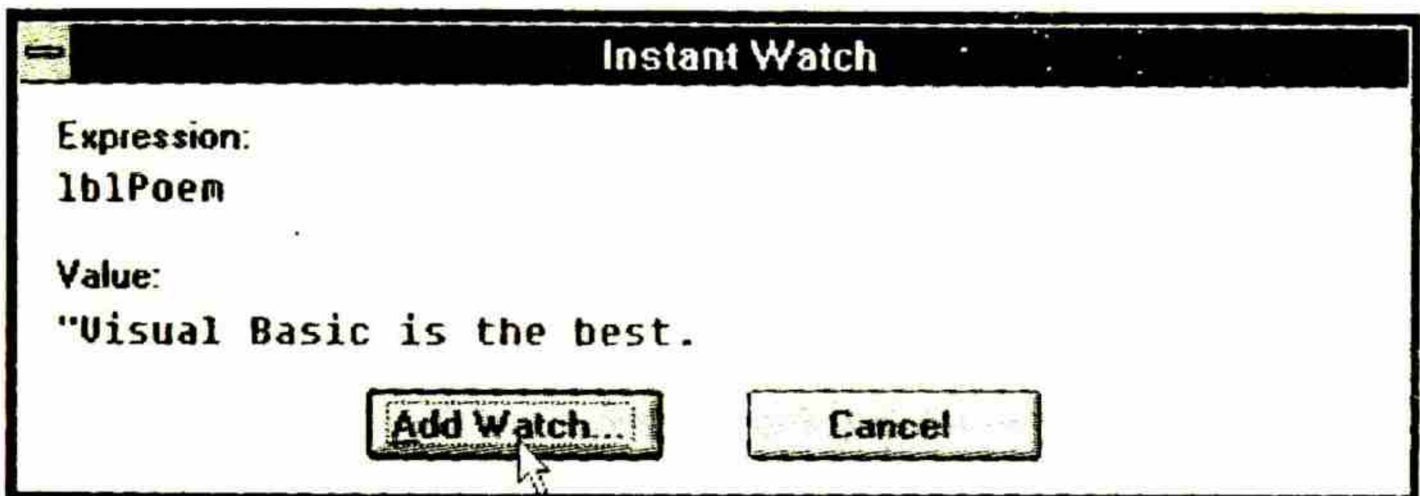
Ứng dụng CHECK2.VBP là chương trình nối và gán một bài thơ vào đề mục của một nhãn. Thủ tục Form_Load() xây dựng bài thơ được nối. Điểm dừng bạn đặt ở dòng lệnh xây dựng phần giữa bài thơ. Hai giá trị duy nhất được khởi tạo tại điểm dừng. Biến chuỗi Newline gồm ký tự xuống dòng và trở về đầu dòng, và điều khiển tên lblPoem sẽ chứa nội dung của bài thơ.

Tại điểm dừng, nhãn lblPoem chứa hai dòng đầu tiên của bài thơ. Hãy nhớ rằng tất cả điều khiển đều có các thuộc tính mặc định, và thuộc tính Caption là thuộc tính mặc định của nhãn. Vì vậy, trong quá trình xây dựng bài thơ, mã lệnh không cần xác định nội dung bài thơ là để nhập vào thuộc tính Caption của nhãn. Như bạn sẽ thấy, bất cứ khi nào bạn chỉ xác định tên nhãn, Visual Basic sẽ xem như bạn muốn gán hay hiển thị thuộc tính Caption.

Hãy thực hiện các bước sau để hiểu loại công việc nào bạn có thể thực hiện tại điểm dừng:

1. Bằng cách kéo mouse, hãy hiện sáng hoặc liệt kê điều khiển lblPoem trên dòng lệnh tại điểm dừng.
2. Nhấp biểu tượng kính lúp trên thanh công cụ. Nút lệnh kính lúp trên thanh công cụ sẽ đưa ra hộp thoại Instant Watch (có hiệu lực trong menu Debug). Visual Basic sẽ hiển thị dòng lệnh đầu tiên được lưu trong thuộc tính Caption của lblPoem, như minh họa trong Hình 24.4.
3. Lúc đầu, bạn có lẽ thất vọng khi hộp thoại Instant Watch không trình bày hai dòng nội dung bài thơ trong thuộc tính Caption của nhãn. Hãy nhớ rằng một nhãn có thể chứa một đề mục dài hàng ngàn ký tự. Visual Basic có khả năng chỉ cho bạn các ký tự xuống dòng và trở về đầu dòng ở cuối dòng thơ đầu tiên trong đề mục.

4. Nhấp nút lệnh Cancel của hộp thoại Instant Watch để đóng hộp thoại. Visual Basic sẽ trở lại điểm dừng của cửa sổ Code.
5. Thông thường, lập trình viên sẽ thi hành từng dòng lệnh sau một điểm dừng. Để thi hành một dòng lệnh lúc đó, bạn có thể chọn lệnh Debug Single Step, hay nhấn phím F8, hoặc nhấp biểu tượng bàn chân trên thanh công cụ. Khi bạn thi hành từng bước mã lệnh, Visual Basic sẽ hiện sáng dòng lệnh cần thi hành kế tiếp. Tại thời điểm bất kỳ trong quá trình xử lý từng bước, bạn có thể xem xét các biến và các điều khiển với hộp thoại Instant Watch.



Hình 24.4. *Hiển thị giá trị nhãn.*

Cuối cùng, thủ tục Form_Load() kết thúc và mẫu biểu xuất hiện trên màn hình. Nếu bạn nhấp một nút tùy chọn, Visual Basic sẽ thi hành thủ tục biến cố của nút tùy chọn đó và đồng thời mở lại cửa sổ Code để bạn có thể tiếp tục quá trình xử lý từng bước.

Giả sử một biến có giá trị không đúng nhưng bạn chẳng thể nào xác định chính xác được nơi lỗi xảy ra. Bạn có thể đặt một điểm dừng trên từng dòng mã lệnh thay đổi giá trị biến. Khi chạy chương trình, bạn sẽ thấy nội dung các biến đó trước và sau khi lệnh có điểm dừng. Nếu điểm dừng đầu tiên đã khởi tạo biến đúng, bạn không cần thực hiện từng bước các mã lệnh kế tiếp cho đến khi gặp điểm dừng kế tiếp. Thay vì thi hành từng bước (single stepping), bạn có thể chọn lệnh Run Continue hay nhấn F5 để trả quá trình thi hành về chế độ Runtime bình thường của nó (chế độ thời gian thi hành). Khi Visual Basic gặp điểm dừng kế tiếp, mã lệnh sẽ dừng ở điểm dừng kế tiếp và bạn có thể tiếp tục xem xét giá trị biến.

Bạn có thể mở hộp thoại Add Watch từ menu Debug hay từ hộp thoại Instant Watch. Hộp thoại Add Watch được mô tả trong Hình 24.5. Hộp thoại Add Watch cho phép bạn nhập biểu thức bất kỳ trong hộp nhập Expression.

Add Watch
Expression:
lblPoem
OK
Cancel
Context
Procedure
chkAll_Click
Form/Module
CHECK2.FRM
Global
Watch Type
Watch Expression
Break when Expression is True
Break when Expression has Changed

Hình 24.5. *Khởi tạo các biểu thức mà Visual Basic theo dõi trong hộp thoại Add Watch.*

Hộp thoại Add Watch có ích cho những lỗi dữ liệu mà bạn hầu như không thể tìm thấy trong mã lệnh. Đúng hơn là quá trình dừng tại mỗi dòng lệnh có sử dụng giá trị dữ liệu. Bạn có thể thêm một biến hay biểu thức vào hộp thoại Add Watch, và Visual Basic sẽ dừng quá trình thi hành khi biến đó thay đổi, khi biểu thức bạn nhập trong hộp thoại Add Watch trả về giá trị True, hay khi Visual Basic thay đổi giá trị biểu thức (Frame Watch Type ở cuối hộp thoại Add Watch sẽ xác định cách bạn muốn khởi tạo kiểu xem).

Ví dụ, giả sử một biến đếm số khách hàng, nhưng một số nơi trong mã lệnh của bạn có giá trị âm xuất hiện trong biến. Nếu bạn đã thêm một biểu thức để theo dõi như CustCnt < 0 vào dấu nhắc Expression trong hộp thoại Add Watch, và nhấp Break khi nút tùy chọn Expression là True, sau đó bạn có thể chạy chương trình và Visual Basic sẽ chuyển sang chế độ Break ngay tại dòng lệnh đã làm cho biến trở nên có giá trị âm.

Các nút trên thanh công cụ với biểu tượng hai bàn chân làm việc giống hệt biểu tượng thi hành từng bước ngoại trừ việc Visual Basic sẽ thi hành thủ tục hiện hành của điểm dừng theo chế độ từng bước nhưng sẽ thi hành trọn vẹn tất cả thủ tục được gọi bởi thủ tục đang thi hành từng bước. Kết thúc phần dò lỗi hiện hành của bạn bằng cách chọn lệnh Run End. Visual Basic sẽ trở lại chế độ Design (thiết kế).

Củng cố

Các hộp thoại bạn đòi hỏi trong khi dò lỗi mã lệnh tạo điều kiện cho quá trình phân tích các biến và xem kết quả xác định. Bạn có thể xem nội dung biến và điều khiển để chắc chắn rằng dữ liệu đang được khởi tạo theo cách bạn mong đợi. Hơn nữa, hộp thoại Add Watch còn cho phép bạn khởi tạo các biểu thức mà Visual Basic xem trong suốt quá trình thi hành chương trình. Nếu giá trị các biểu thức này luôn là True hay thay đổi thành True, Visual Basic sẽ dừng tại dòng lệnh đó và cho phép bạn phân tích các giá trị đang dùng trong hộp thoại Instant Watch.

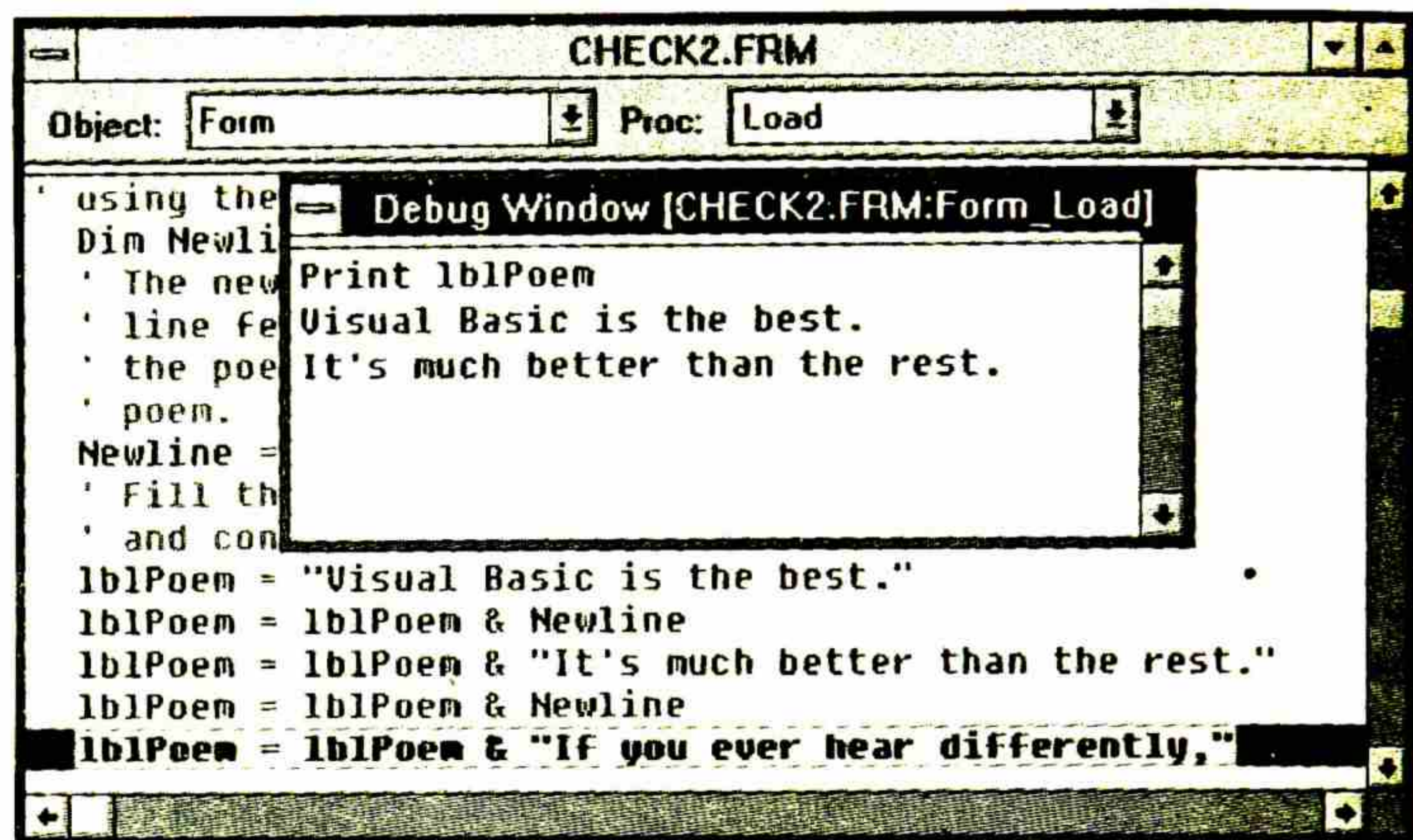
SỬ DỤNG CỬA SỔ IMMEDIATE

Khái niệm

Cửa sổ Immediate cho phép bạn xem cũng như thay đổi biến và điều khiển trong quá trình thi hành một chương trình.

Tại điểm dừng bất kỳ, bạn có thể chọn lệnh Window Debug (Ctrl+B) để yêu cầu cửa sổ Debug. Cửa sổ Debug là một cửa sổ đặc biệt, bạn có thể nhập trực tiếp các lệnh Visual Basic, xem và thay đổi giá trị các biến và điều khiển trong quá trình thi hành một chương trình.

Muốn in, bạn có thể áp dụng phương thức Print (xem Bài 21) để xem biến và điều khiển. Khi bạn sử dụng phương thức Print trong cửa sổ Debug, Visual Basic sẽ gửi kết quả ra cửa sổ Debug chứ không phải máy in, như trong Bài 21. Ví dụ, giả sử bạn đặt một điểm dừng trong quá trình nối bài thơ, được mô tả ở phần trước, và nhấn Ctrl+B để mở cửa sổ Debug. Cửa sổ Debug nhận ra những lệnh và phương thức đơn giản như Print chẳng hạn và các lệnh gán.



Hình 24.6. *Trong cửa sổ Debug, bạn có thể in giá trị biến và điều khiển.*

Hình 24.6 mô tả những gì sẽ xảy ra nếu in giá trị của lblPoem. Không giống hộp thoại Instant Watch, cửa sổ Debug sẽ hiển thị toàn bộ giá trị nhãn trên nhiều dòng. Bạn có thể định lại kích cỡ và di chuyển cửa sổ Debug. Mặc dù phải dùng lệnh Print thay vì chỉ nhấp một biến hay điều khiển, nhiều người dùng thích hiển thị các giá trị từ cửa sổ Debug hơn là hộp thoại Instant Watch bởi vì cửa sổ Debug sẽ hiển thị toàn bộ giá trị và có thanh cuốn dọc để người dùng có thể cuộn qua các giá trị được in trong cửa sổ.

Hãy dùng cửa sổ Debug bên trong mã lệnh của bạn: Đặc điểm cuộn và định lại kích cỡ của cửa sổ Debug đôi khi khá thuận lợi cho các lập trình viên thích gửi thông báo vào cửa sổ Debug lúc chạy chương trình hơn là sử dụng hộp thoại Instant Watch. Ví dụ, nếu bạn muốn xem giá trị các đối số xác định khi thủ tục được gọi thi hành, bạn có thể thêm lệnh Print ở đầu các thủ tục để tự gửi các giá trị đối số đến cửa sổ Debug khi chương trình thi hành. Một khi bạn sửa xong lỗi kỹ thuật trong chương trình, bạn có thể gỡ bỏ các lệnh Print để cửa sổ Debug đóng lại.

Để in đến cửa sổ Debug, hãy bắt đầu phương thức Print trong đối tượng Debug. Lệnh sau được thi hành bất cứ nơi đâu từ cửa sổ Code của trình ứng dụng, sẽ in các giá trị của 2 biến tiêu đề thích hợp trong cửa sổ Debug:

```
Debug.Print "Age:"; ageVal, "Weight:"; weightVal
```

Tất cả tùy chọn của phương thức Print, bao gồm cả dấu ;, dấu ,, hàm Tab() và Spc(), sẽ làm việc trong cửa sổ Debug như khi chúng thực hiện với đối tượng Printer trong Bài 21. Tuy nhiên, hãy cẩn thận xác định đối tượng Debug trước phương thức Print, nếu bạn bỏ quên Debug, Visual Basic sẽ in kết quả trực tiếp ra mẫu biểu!

Cửa sổ Debug nhận biết các phép gán mà bạn thực hiện cho các biến và điều khiển. Ví dụ, giả sử bạn biết rằng một biến xác định đã không được khởi tạo đúng trước đó, nhưng bạn vẫn muốn hoàn thành quá trình thi hành chương trình. Nếu cần bạn có thể gán trực tiếp cho biến một giá trị mới trong cửa sổ Debug bằng cách dùng lệnh gán. Khi bạn chuyển quá trình thi hành chương trình vào hoặc ở chế độ từng bước hoặc ở chế độ Runtime, chương trình từ thời điểm đó sẽ hiểu biến có giá trị bạn đã gán cho nó.

Củng cố

Cửa sổ Debug cho bạn nhiều điều khiển hơn để hiển thị các biến và điều khiển. Cửa sổ Debug sẽ tự điều chỉnh kích cỡ và cung cấp cho bạn một thanh cuốn để bạn có thể thấy nội dung tất cả các biến và điều khiển và có thể cuộn những giá trị này khi bạn dò lỗi trong lúc chương trình thi hành. Cửa sổ Debug cũng cho bạn khả năng gán các giá trị cho biến và điều khiển. Giữa quá trình thi hành chương trình, bạn có thể chuyển sang chế độ dừng và thay đổi các giá trị bất kỳ. Khi bạn tiếp tục quá trình thi hành chương trình, chương trình sẽ dùng những giá trị bạn đã gán trong cửa sổ Debug.

Bài tập

Kiến thức tổng quát

1. Hai loại lỗi chính mà chương trình có thể gặp là gì?
2. Các lỗi văn phạm và đánh vần thuộc loại lỗi nào?
3. *Trình gỡ rối* (debugger) là gì?
4. Cách bạn có thể yêu cầu Visual Basic kiểm tra các lỗi cú pháp khi nhập chương trình vào trong cửa sổ Code của Visual Basic?

5. Đúng hay Sai: Visual Basic sẽ hiển thị hộp thông báo khi một lỗi cú pháp xuất hiện.
6. Đúng hay Sai: Visual Basic sẽ hiển thị hộp thông báo khi một lỗi logic xuất hiện.
7. Loại lỗi nào là khó tìm nhất, lỗi cú pháp hay lỗi logic?
8. Chương trình có thể tồn tại trong ba chế độ nào?
9. Visual Basic ở trong chế độ nào khi bạn viết chương trình và đặt các mục trên mẫu biểu?
10. Làm cách nào để biết chế độ hiện hành của chương trình là gì?
11. Chế độ nào là hoàn hảo cho quá trình hiển thị và thay đổi các biến?
12. Tên hai cách chuyển sang chế độ dừng.
13. Điểm dừng là gì?
14. Bạn có thể đặt một điểm dừng như thế nào?
15. Do đâu bạn có thể biết một điểm dừng được đặt khi bạn nhìn vào cửa sổ Code?
16. Điều gì sẽ xảy ra khi Visual Basic gặp một điểm dừng trong quá trình thi hành chương trình?
17. Nghĩa của *thi hành từng bước* (single tepping) trong chương trình là gì?
18. Điểm khác biệt giữa nút single footprint và double footprint trên thanh công cụ?
19. Điểm khác biệt giữa hộp thoại Instant Watch với hộp thoại Add Watch?
20. Loại quan sát nào là có ích nhất cho quá trình hiển thị các biến tạo điểm dừng chương trình?
21. Cửa sổ nào cho phép bạn xem và thay đổi biến và điều khiển Visual Basic trong thời gian chạy chương trình?
22. Lệnh nào cho phép bạn hiển thị các giá trị dữ liệu trong cửa sổ Debug?
23. Lệnh nào cho phép bạn gán các giá trị mới vào biến và điều khiển trong thời gian chạy chương trình?

Tìm lỗi kỹ thuật

24. An Huy muốn gửi một số giá trị vào cửa sổ Debug để duy trì danh sách giá trị biến có thể cuộn khi chạy chương trình. Sau đây là một số lệnh mà An Huy đang cố đưa vào cửa sổ Debug:

```
Print lblTitle.Caption
Print x, y, z
Print CompName
```

Ngay cả khi An Huy mở cửa sổ Debug trong quá trình thi hành chương trình, cửa sổ trống. Thay vì chuyển tới cửa sổ Debug, các phương thức Print của An Huy dường như xuất hiện trên mẫu biểu. Hãy giúp anh ấy giải quyết vấn đề.

Bài thực hành 12

Hướng ra bên ngoài

Tóm tắt

Chương này chỉ bạn cách dùng thanh cuốn và điều khiển khung lưới (Grid) để giám sát mouse, và sử dụng trình gỡ rối trong Visual Basic. Thanh cuốn là một điều khiển linh hoạt giúp người dùng trong quá trình nhập và chọn từ một vùng giá trị. Khung lưới cho phép người dùng xem bảng thông tin ở dạng giống bảng tính. Khi cần giám sát quá trình di chuyển hay nhấp mouse của người dùng, bạn có thể viết lệnh bổ sung để đáp ứng các thao tác đó. Visual Basic hỗ trợ quá trình thi hành các mã lệnh này cho tất cả điều khiển.

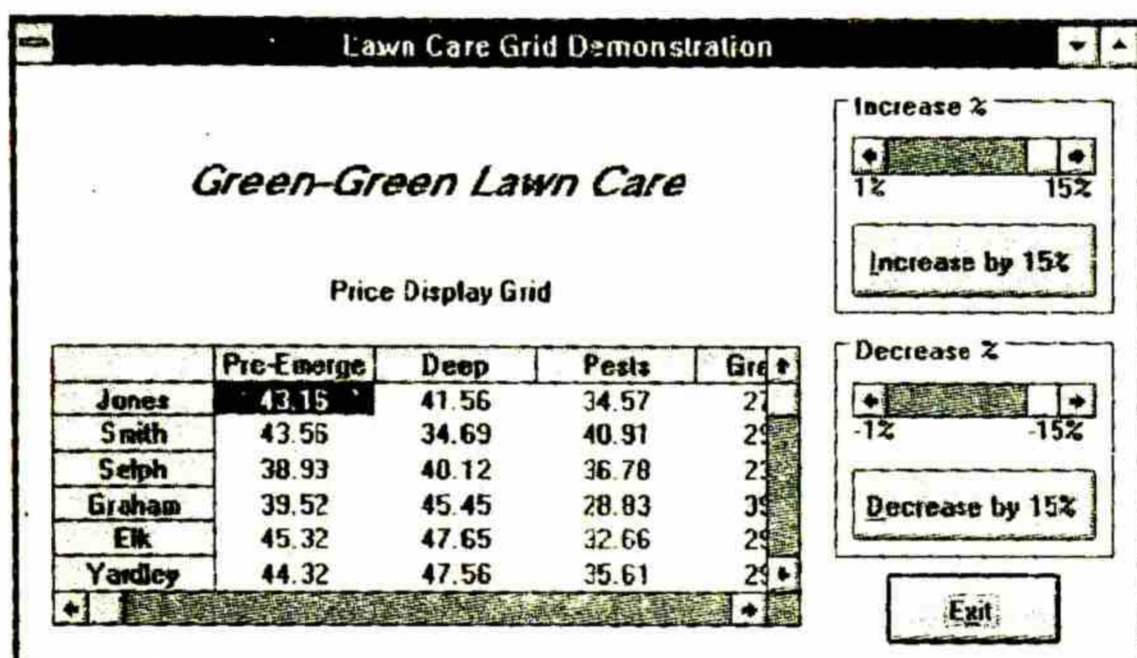
Trình gỡ rối là công cụ phát triển tích hợp giúp bạn dò tìm lỗi trong mã lệnh chương trình. Trình gỡ rối cho phép bạn thực hiện từng bước mã lệnh chương trình, giám sát biến, và kiểm tra giá trị dữ liệu chương trình tại một dòng mã lệnh bất kỳ trong quá trình thi hành chương trình.

Trong chương này, bạn đã tìm hiểu :

- Cách thiết đặt và giám sát điều khiển thanh cuốn
- Cách khởi tạo khung lưới cho các bảng dữ liệu
- Cách xác định các ô được chọn trong khung lưới
- Cách giám sát quá trình di chuyển mouse của người dùng
- Cách sử dụng trình gỡ rối được tích hợp trong Visual Basic để những các lỗi kỹ thuật khó hiểu trong chương trình

Mô tả chương trình

Hình P12.1 minh họa cửa sổ Form đang mở PROJECT12.VBP. Trước tiên người dùng sẽ thấy bảng giá trị được định dạng lưu trong một điều khiển khung lưới.



Hình P12.1. *Ứng dụng Project12 khởi đầu với một mẫu biểu đơn giản.*

Lưu ý

Bạn phải chắc chắn điều khiển khung lưới đã được bổ sung vào thanh công cụ của Visual Basic. Xem Bài 23 để biết thêm chi tiết về quá trình thêm tập tin này.

Trình ứng dụng sẽ minh họa cách dùng, thay đổi và giám sát công dụng của điều khiển khung lưới đối với người dùng, cũng như cách bạn có thể dùng thanh cuốn để thay đổi hành vi các điều khiển khác trên mẫu biểu.

NGHIÊN CỨU ĐIỀU KHIỂN KHUNG LƯỚI

Điều khiển khung lưới chứa bảng giá thuê bãi của 8 khách hàng ứng với 5 loại bãi. Các bãi này cần 40 dịch vụ thuê trong năm. Khi công ty thêm khách hàng mới, điều khiển khung lưới có thể dễ dàng mở rộng bằng cách thay đổi giá trị thuộc tính Grid và thêm giá trị mới vào mã lệnh. Một khi số khách hàng vượt quá số quy định, chương trình muốn liên kết với ổ đĩa và lưu trữ bảng giá hiện hành lên đĩa để dễ dàng bảo trì đồng thời các chương trình khác có thể sử dụng.

Hãy thử điều này: Nhấp thanh cuốn trong điều khiển khung lưới để cuộn bảng giá và xem 40 giá. Trước khi suy nghĩ về các nút lệnh và thanh cuốn ngang ở bên phải màn hình, hãy nghiên cứu Ví dụ P12.1 để nắm được mục đích tổng quát của chương trình.

Ví dụ P12.1. *Thủ tục Form_Load() điều khiển quá trình vận hành chương trình ban đầu.*

```
1: Sub Form_Load ()
2: ' Defines the justification of the cells
3: ' as well as assigns cell titles to the
4: ' fixed row and columns and assigns initial
5: ' price values to the 40 cells.
6: Call InitScrolls ' Initialize scroll bars
7: Call CenterCells ' Center all cell alignments
8: Call SizeCells ' Specify width of cells
9: Call Titles ' Initialize column and row titles
10: Call FillCells ' Fill cells with values
11: End Sub
```

(6: Chia các module dài thành nhiều module ngắn hơn.)

Khái niệm mới

Chương trình được cấu trúc là chương trình có mã lệnh chia thành nhiều phần nhỏ.

Phân tích

Mã lệnh trong Ví dụ P12.1 minh họa phương pháp lập trình gọi là lập trình có cấu trúc mà bạn nên kết hợp trong chương trình riêng của mình. Form_Load() không phải là thủ tục biến cố lớn bởi vì các lệnh Call từ dòng 6 đến dòng 10 sẽ giúp chia mã lệnh thành nhiều thủ tục con nhỏ hơn và dễ quản lý hơn. Tất cả mã lệnh được gọi sẽ thi hành khi thủ tục biến cố Form_Load() thi hành (ngay trước khi người dùng thấy mẫu biểu trên màn hình), nhưng 5 thủ tục con cung cấp một cách chia chương trình thành nhiều phần để khi cần thiết bạn quyết định sửa đổi hoàn toàn một phần nào đó.

Giả sử bạn cần cập nhật thông tin về giá. Nếu chỉ nghiên cứu thủ tục Form_Load(), bạn có thể thấy thủ tục con FillCells sẽ điền giá trị vào các ô trong điều khiển khung lưới. Vì vậy, bạn có thể đi thẳng tới thủ tục đó nếu bạn cần thay đổi về giá.

Ví dụ P12.2 trình bày mã lệnh các thủ tục được gọi từ thủ tục Form_Load(). Hãy nghiên cứu mã lệnh để làm quen với những yêu cầu khởi tạo điều khiển khung lưới.

Lưu ý

Không phải tất cả mã lệnh thủ tục con FillCells và Tiltes được liệt kê trong Ví dụ P12.2 bởi vì mã lệnh dài và lặp đi lặp lại. Chuỗi lệnh gán dài mô tả cách nạp và lưu thông tin giá vào đĩa sẽ có ích khi công ty thêm nhiều khách hàng. Bạn không muốn xem và sửa đổi các lệnh gán chuỗi dài mỗi lần một giá cần cập nhật, thêm một khách hàng mới, hay xóa một khách hàng. Tuy nhiên, quá trình sử dụng đĩa lưu bảng giá sẽ đòi hỏi ít nhất một chương trình phụ cho phép bạn cập nhật trực tiếp thông tin giá trên đĩa.

Ví dụ P12.2. *Mã lệnh thủ tục Form_Load() sẽ gọi để khởi tạo điều khiển khung lưới và các điều khiển khác.*

```
Sub InitScrolls ()
' Set both scroll bars to their maximum values
hscIncrease.Value = 15
hscDecrease.Value = 15
End Sub
Sub CenterCells ()
' Sets the justification of the grid's
' cells to center alignment
Dim Apps As Integer ' 5 applications yearly
' Center all values in the grid
For Apps = 0 To 5 ' 5 applications
grdLawn.Col = Apps ' Locate next column
' Center the fixed cells in this column
grdLawn.FixedAlignment(Apps) = 2
Next Apps
For Apps = 1 To 5 ' 5 applications
' Center the non-fixed cells in this column
grdLawn.ColAlignment(Apps) = 2
Next Apps
End Sub
```

```
Sub SizeCells ()
' Specify the width of each cell
Dim Apps As Integer ' 5 Application
For Apps = 0 To 5
grdLawn.ColWidth(Apps) = 1100 ' Twips
Next Apps
End Sub

Sub Titles ()
' Fill in the column titles
grdLawn.Row = 0 ' All titles are in row #0
grdLawn.Col = 1
grdLawn.Text = "Pre-Emerge"
grdLawn.Col = 2
grdLawn.Text = "Deep"
' Rest of column titles are filled here
' Fill in the row titles
grdLawn.Col = 0 ' Customer titles in first column
grdLawn.Row = 1
grdLawn.Text = "Jones"
grdLawn.Row = 2
grdLawn.Text = "Smith"
' Rest of row titles are filled here
End Sub

Sub FillCells ()
' Fill in all 40 cells one at a time.
' This is tedious!
'
' Normally, you would fill these cells from
' a disk file or through calculations.
grdLawn.Row = 1
grdLawn.Col = 1
```

```
56: grdLawn.Text = 43.16
57: grdLawn.Col = 2
58: grdLawn.Text = 41.56
59: grdLawn.Col = 3
60: grdLawn.Text = 34.57
61: grdLawn.Col = 4
62: grdLawn.Text = 27.49
63: grdLawn.Col = 5
64: grdLawn.Text = 41.34
65: grdLawn.Row = 2
66: grdLawn.Col = 1
67: grdLawn.Text = 43.56
68: ' Rest of cells filled here
69: End Sub
```

Mô tả

1: Mã lệnh của thủ tục Subroutine đặt thanh cuốn đến giá trị tối đa ban đầu của nó.

2: Chú thích giải thích thủ tục.

3: Đặt thanh cuốn Increase đến giá trị tối đa bằng 15, cho biết sẽ tăng 15% giá trị khi người dùng nhấp nút lệnh Increase.

4: Đặt thanh cuốn Decrease đến giá trị tối đa bằng 15, cho biết sẽ giảm 15% giá trị khi người dùng nhấp nút lệnh Decrease.

5: Kết thúc thủ tục con.

6: Dòng trống phân tách thủ tục.

7: Mã lệnh thủ tục Subroutine đặt giá trị canh giữa cho thuộc tính canh lề Alignment trong các ô điều khiển khung lưới.

8: Chú thích giải thích thủ tục.

9: Tiếp tục chú thích trên.

10: Định nghĩa một biến dùng để lặp số trong trình ứng dụng.

11: Chú thích giải thích mã lệnh.

12: Duyệt qua 6 cột trong bảng (cột đầu tiên là cột 0, lưu tiêu đề hàng).

13: Gán số cột cho giá trị vòng lặp For.

14: Chú thích giải thích mã lệnh.

15: Thuộc tính canh giữa (Giá trị 2 sẽ đặt thuộc tính thành 2–Center) cho tất cả các hàng cố định trong cột được chọn. Chỉ hàng đầu tiên được cố định trong 5 cột cuối cùng.

(15: Visual Basic sẽ canh phải các giá trị nếu bạn không thay đổi thuộc tính canh lề.)

16: Tiếp tục vòng lặp.

17: Duyệt qua 5 cột cuối trong bảng (các cột này là những cột không cố định) để sửa soạn canh giữa.

18: Chú thích giải thích mã lệnh.

19: Thuộc tính canh giữa (Giá trị 2 sẽ đặt thuộc tính thành 2–Center) cho tất cả các hàng không cố định trong cột được chọn.

20: Tiếp tục vòng lặp.

21: Kết thúc thủ tục.

22: Dòng trống phân tách các thủ tục.

23: Mã lệnh của thủ tục Subroutine định độ rộng các ô.

24: Chú thích giải thích thủ tục.

25: Định nghĩa biến được dùng để lặp qua một số lần trong trình ứng dụng.

26: Duyệt qua từng cột, kể cả cột 0 cố định đầu tiên.

27: Định độ rộng từng ô bằng 1.100 twip.

(27: Phải đảm bảo độ rộng mỗi ô đủ để lưu tất cả dữ liệu và tiêu đề.)

28: Tiếp tục vòng lặp.

29: Kết thúc thủ tục con.

30: Dòng trống phân tách thủ tục.

31: Mã lệnh thủ tục Subroutine khởi tạo tất cả tiêu đề trong các hàng và cột cố định của khung lưới.

32: Chú thích giải thích thủ tục.

33: Khởi tạo hàng đầu tiên (hàng số 0).

34: Khởi tạo cột thứ hai (cột số 1).

35: Gán tiêu đề.

36: Di chuyển đến cột kế tiếp trong cùng hàng.

37: Gán tiêu đề.

38: Chú thích giải thích mã lệnh không được mô tả.

39: Chú thích giải thích mã lệnh.

40: Khởi tạo cột đầu tiên (cột số 0).

41: Khởi tạo hàng thứ hai (hàng số 1).

42: Gán tiêu đề.

43: Di chuyển tới hàng kế tiếp trong cùng cột.

44: Gán tiêu đề.

45: Chú thích giải thích mã lệnh không được mô tả.

46: Kết thúc thủ tục con.

47: Dòng trống phân tách các thủ tục.

48: Mã lệnh của thủ tục Subroutine khởi tạo giá trị các ô không cố định trong điều khiển khung lưới.

49: Chú thích giải thích thủ tục.

50: Tiếp tục chú thích.

51: Tiếp tục chú thích.

52: Tiếp tục chú thích.

53: Tiếp tục chú thích.

54: Khởi tạo hàng thứ hai (hàng không cố định đầu tiên).

(54: Thuộc tính Row và Col xác định ô nào nhận giá trị kế tiếp.)

55: Khởi tạo cột thứ hai (cột không cố định đầu tiên).

56: Gán giá trị giá.

57: Di chuyển đến cột kế tiếp.

58: Gán giá trị giá.

59: Di chuyển đến cột kế tiếp.

60: Gán giá trị giá.

61: Di chuyển đến cột kế tiếp.

62: Gán giá trị giá.

63: Di chuyển đến cột kế tiếp.

64: Gán giá trị giá.

65: Di chuyển đến hàng kế tiếp.

66: Di chuyển đến cột kế tiếp.

67: Gán giá trị giá.

68: Chú thích giải thích mã lệnh không được liệt kê ở đây.

69: Kết thúc thủ tục con.

DÙNG NÚT LỆNH VÀ THANH CUỐN

Trình ứng dụng được khởi tạo cho phép người dùng chọn một hay nhiều ô bằng mouse và sau đó cập nhật giá chỉ với các ô được chọn. Do đó, người dùng có thể thử các bảng giá khác nhau để thấy cấu trúc giá làm việc tốt với trình ứng dụng xác định. Khi người dùng nhấp nút lệnh, tất cả ô được chọn sẽ tăng hoặc giảm giá trị tương ứng với nút lệnh được nhấp vào thời điểm đó. Tất cả ô không được chọn vẫn giữ nguyên giá trị cũ.

Thanh cuốn sẽ ảnh hưởng đến việc thay đổi tỉ lệ phần trăm giá trên đề mục nút lệnh. Chẳng hạn, nếu người dùng giảm giá trị của thanh cuốn Decrease, đề mục của nút lệnh Decrease by... sẽ thay đổi theo. Lần kế tiếp người dùng nhấp nút lệnh, ô được chọn sẽ giảm giá ứng với số được chỉ trên thanh cuốn. Mã lệnh trong Ví dụ P12.3 sẽ mô tả mối liên hệ giữa 2 thanh cuốn và các nút lệnh thích hợp của chúng.

Ví dụ P12.3. *Quá trình quản lý giá thay đổi qua thanh cuốn, nút lệnh và các ô được chọn.*

```
Sub hscDecrease_Change ()
' Change the command button's caption
cmdDecrease.Caption = "&Decrease by"
        & Str$(hscDecrease.Value) & "%"
End Sub

Sub hscIncrease_Change ()
' Change the command button's caption
cmdIncrease.Caption = "&Increase by"
        & Str$(hscIncrease.Value) & "%"
End Sub

Sub cmdDecrease_Click ()
' Decrease selected cell values
' by the decreasing scroll bar percentage
Dim SelRows, SelCols As Integer
If (grdLawn.HighLight) Then ' If true...
```

```
For SelRows = grdLawn.SelStartRow To grdLawn.SelEndRow
For SelCols = grdLawn.SelStartCol To grdLawn.SelEndCol
grdLawn.Row = SelRows
grdLawn.Col = SelCols
' Decrease the cell by scroll bar amount
grdLawn.Text = grdLawn.Text
    – (hscDecrease.Value / 100 * grdLawn.Text)
grdLawn.Text = Format(grdLawn.Text, "Fixed")
Next SelCols
Next SelRows
End If
End Sub

Sub cmdIncrease_Click ()
' Increase selected cell values
' by the amount of the increase scroll bar's percentage
Dim SelRows, SelCols As Integer
If (grdLawn.HighLight) Then ' If true...
For SelRows = grdLawn.SelStartRow To grdLawn.SelEndRow
For SelCols = grdLawn.SelStartCol To grdLawn.SelEndCol
grdLawn.Row = SelRows
grdLawn.Col = SelCols
' Increase the cell by scroll bar amount
grdLawn.Text = grdLawn.Text * (1 + hscIncrease.Value / 100)
grdLawn.Text = Format(grdLawn.Text, "Fixed")
Next SelCols
Next SelRows
End If
End Sub

Sub cmdExit_Click ()
End
End Sub
```

Mô tả

1: Mã lệnh thủ tục biến cố sẽ thi hành khi người dùng thay đổi thanh cuốn giảm.

2: Chú thích giải thích thủ tục.

3: Thay đổi đề mục của nút lệnh để phản ánh phần trăm mức giảm mới.

3: Một thủ tục biến cố có thể thay đổi giá trị thuộc tính trong thủ tục biến cố khác.

4: Kết thúc thủ tục.

5: Dòng trống phân tách các thủ tục

6: Mã lệnh thủ tục biến cố sẽ thi hành khi người dùng thay đổi thanh cuốn tăng.

7: Chú thích giải thích thủ tục.

8: Thay đổi đề mục của nút lệnh để phản ánh phần trăm mức tăng mới.

9: Kết thúc thủ tục.

10: Dòng trống phân tách các thủ tục

11: Mã lệnh thủ tục biến cố sẽ thi hành khi người dùng nhấp nút lệnh Decrease.

12: Chú thích giải thích thủ tục biến cố.

13: Tiếp tục chú thích.

14: Định nghĩa hai biến điều khiển các ô được chọn.

15: Nếu có các ô được hiện sáng...

(15: Không có ô được chọn nếu không có ô nào được hiện sáng.)

16: Duyệt qua các hàng được chọn.

17: Duyệt qua các cột được chọn.

18: Định hàng để thay đổi.

19: Định cột để thay đổi.

20: Chú thích giải thích mã lệnh.

21: Cập nhật giá trị ô với tổng số giảm.

22: Định dạng ô với 2 vị trí thập phân.

23: Tiếp tục duyệt qua các cột được chọn.

24: Tiếp tục duyệt qua các hàng được chọn.

25: Kết thúc lệnh If.

26: Kết thúc thủ tục.

27: Dòng trống phân tách các thủ tục.

28: Mã lệnh thủ tục biến cố sẽ thi hành khi người dùng nhấp nút lệnh Increase.

29: Chú thích giải thích thủ tục biến cố.

30: Tiếp tục chú thích.

31: Định nghĩa hai biến điều khiển các ô được chọn.

32: Nếu có các ô được hiện sáng...

33: Duyệt qua các hàng được chọn.

34: Duyệt qua các cột được chọn.

35: Định hàng được thay đổi.

36: Định cột được thay đổi.

37: Chú thích giải thích mã lệnh.

38: Cập nhật giá trị ô với tổng số tăng.

(38: Không ảnh hưởng đến các ô không được chọn.)

39: Định dạng ô với hai vị trí thập phân.

40: Tiếp tục duyệt qua các ô được chọn.

41: Tiếp tục duyệt qua các hàng được chọn.

42: Kết thúc lệnh If.

43: Kết thúc thủ tục.

44: Dòng trống phân tách các thủ tục.

45: Mã lệnh thủ tục biến cố sẽ thi hành khi người dùng nhấp nút lệnh Exit.

46: Kết thúc chương trình.

47: Kết thúc thủ tục.

Đóng trình ứng dụng

Bạn có thể thoát khỏi trình ứng dụng và Visual Basic. Xin chúc mừng! Giờ đây bạn đã là một lập trình viên Visual Basic rồi đấy!

Phụ lục A

Độ ưu tiên toán tử

Có 3 tập hợp toán tử, các toán tử số học, so sánh, và logic. Ba bảng phụ lục này sẽ mô tả thứ tự ưu tiên của từng tập hợp toán tử.

Bảng A.1. *Độ ưu tiên của toán tử số.*

Mức	Toán tử
1	Lũy thừa (^)
2	Số âm (-)
3	Nhân và chia (*, /)
4	Chia nguyên (\)
5	Chia lấy phần dư (Mod)
6	Cộng và trừ (+, -)
7	Nối chuỗi (&)

Bảng A.2. *Độ ưu tiên của toán tử so sánh*

Mức	Toán tử
1	Bằng (=)
2	Khác (<>)
3	Nhỏ hơn (<)
4	Lớn hơn (>)
5	Nhỏ hơn hoặc bằng (<=)
6	Lớn hơn hoặc bằng (>=)
7	Like

Bảng A.3. *Độ ưu tiên của toán tử logic*

Mức	Toán tử
1	Not
2	And
3	Or
4	Xor
5	Eqv
6	Imp
7	Is

Phụ lục B

Các từ khóa

Visual Basic bao gồm một danh sách dài các lệnh, hàm và phương thức. Danh sách này được hiểu như là các từ khóa. Bạn phải cẩn thận để không gán những tên này cho thủ tục, điều khiển hay biến. Nếu không Visual Basic sẽ phát sinh lỗi.

Bảng B.1. *Các từ khóa*

Abs
Access
AddItem
AddNew
Alias
And
Any
App
AppActivate
Append
AppendChunk
Arrange
As
Asc
Atn
Base
Beep
BeginTrans
Binary
ByVal
Call
Case
CCur
CDbl
ChDir
ChDrive
Chr
Chr$
CInt
Circle
Clear
Clipboard
CLng
Close
Cls
Command
Command$
CommitTrans
Compare
Const
Control
Controls
Cos
CreateDynaset
CSng
CStr
CurDir$
Currency
CVar
CVDate
Data
Date
Date$
DateSerial
DateValue
Day
Debug
Declare
DefCur
CefDbl
DefInt
DefLng
DefSng
DefStr
DefVar
Delete

Dim
Dir
Dir$
Do
DoEvents
Double
Drag
Dynaset
Edit
Else
ElseIf
End
EndDoc
EndIf
Environ$
EOF
Eqv
Erase
Erl
Err
Error
Error$
ExecuteSQL
Exit
Exp
Explicit
False
FieldSize
FileAttr
FileCopy
FileDateTime
FileLen
Fix
For
Form
Format
Format$
Forms
FreeFile
Function
Get
GetAttr
GetChunk
GetData
DetFormat
GetText
Global
GoSub
GoTo
Hex
Hex$
Hide
Hour
If
Imp
Input
Input$
InputBox
InputBox$
InStr
Int
Integer
Is
IsDate
IsEmpty
IsNull
IsNumeric
Kill
LBound
LCase
LCase$
Left
Left$
Len
Let
Lib
Like
Line
LinkExecute
LinkPoke
LinkRequest
LinkSend
Load
LoadPicture
Loc
Local
Lock
LOF
Log
Long
Loop
LSet
LTrim
LTrim$
Me
Mid
Mid$
Minute
MkDir

Mod	QBColor	Shared
Month	Random	Shell
Move	Randomize	Show
MoveFirst	Read	Sin
MoveLast	ReDim	Single
MoveNext	Refresh	Space
MovePrevious	RegisterDataBase	Space$
MoveRelative	Rem	Spc
MsgBox	RemoveItem	Sqr
Name	Reset	Static
New	Restore	Step
NewPage	Resume	Stop
Next	Return	Str
NextBlock	RGB	Str$
Not	Right	StrComp
Nothing	Right$	String
Now	RmDir	String$
Null	Rnd	Sub
Oct	Rollback	System
Oct$	RSet	Tab
On	RTrim	Tan
Open	RTrim$	Text
OpenDataBase	SavePicture	TextHeight
Option	Scale	TextWidth
Or	Second	Then
Output	Seek	Time
Point	Select	Time$
Preserve	SendKeys	Timer
Print	Set	TimeSerial
Printer	SetAttr	TimeValue
PrintForm	SetData	To
Private	SetFocus	Trim
PSet	SetText	Trim$
Put	Sgn	True

Type	Until	Wend
TypeOf	Update	While
UBound	Using	Width
UCase	Val	Write
UCase$	Variant	Xor
Unload	VarType	Year
Unlock	Weekday	ZOrder

Phụ lục C

Thiết kế ứng dụng CONTROLS.VBP

Để thiết kế ứng dụng Controls.vbp được mô tả trong Chương 2 – Bài 3, bạn hãy thực hiện các bước sau :

1. Khởi động Visual Basic 6.0.
2. Trong hộp thoại New Project, chọn mục New, sau đó nhấp đúp biểu tượng Standard EXE.
3. Chọn lệnh File ➡ Save Project để mở hộp thoại Save File As, nhập nội dung như sau File name: Controls, Save as Type: Form files (*.frm), nhấp nút lệnh Save.
4. Hộp thoại Save Project file hiển thị, nhập nội dung như sau File name: Controls, Save as Type : Project files (*.vbp), nhấp nút lệnh Save.
5. Chọn lệnh View ➡ Object trên thanh menu Visual Bassic để mở cửa sổ Form.
6. Chọn lệnh View ➡ Properties Window để mở cửa sổ Properties.
7. Nhấp đúp điều khiển nhãn (Label) (biểu tượng chữ A hoa) trên thanh công cụ, một nhãn sẽ được chèn vào giữa mẫu biểu, với tên và đề mục mặc định là Label1.
8. Chọn nhãn, và thay đổi thuộc tính Caption của nó trong cửa sổ Properties thành **Have Fun with Controls!**. Đổi thuộc tính Name thành lblLabel1. Đặt thuộc tính Alignment của nhãn thành 2–Center. Nhấp nút ... sau thuộc tính Font, đặt kích cỡ phông chữ bằng 18. Điều chỉnh kích cỡ nhãn để được như Hình 3.3.
9. Nhấp đúp điều khiển nút lệnh (Command Button) trên thanh công cụ để chèn nút lệnh vào giữa mẫu biểu, thay đổi thuộc tính Caption của nút lệnh thành E&xit, đổi thuộc tính Name thành cmdExit, dời nút lệnh tới vị trí thích hợp trên mẫu biểu. Nhấp đúp trên nút lệnh để mở cửa sổ Code và nhập vào lệnh sau:

```
Private Sub cmdExit_Click()
  End
End Sub
```

10. Đầu mục (General) - (Declarations), nhập vào 2 dòng lệnh sau :

```
Option Explicit
Public dem As Integer
```

11. Chọn lệnh View ➡ Object để quay lại cửa sổ Form. Nhấp đúp điều khiển nút lệnh trên thanh công cụ để chèn nút lệnh thứ hai vào mẫu biểu. Thay đổi thuộc tính Caption của nút lệnh thành &Next control, đổi thuộc tính Name của nút lệnh thành cmdNext, dời nút lệnh tới vị trí thích hợp trên mẫu biểu. Nhấp đúp trên nút lệnh để bật cửa sổ Code và nhập vào những lệnh sau :

```
Private Sub cmdNext_Click()
Select Case dem
    Case 0:
      lblLabel2.Caption = "Label control"
      lblLabel2.Visible = True
      lblLabel3.Visible = True
    Case 1:
      lblLabel3.Visible = False
      txtText1.Visible = True
    Case 2:
      txtText1.Visible = False
      lblLabel2.Caption = "Command button"
      cmdPress.Visible = True
    Case 3:
      cmdPress.Visible = False
      lblLabel2.Caption = "Check Box Control"
      chkCheck1.Visible = True
      chkCheck2.Visible = True
      chkCheck3.Visible = True
    Case 4:
      chkCheck1.Visible = False
      chkCheck2.Visible = False
```

```
    chkCheck3.Visible = False
    lblLabel2.Caption = "Option button Control"
    optOption1.Visible = True
    optOption2.Visible = True
    optOption3.Visible = True
    optOption4.Visible = True
  Case 5:
    optOption1.Visible = False
    optOption2.Visible = False
    optOption3.Visible = False
    optOption4.Visible = False
    lblLabel2.Caption = "Frame Control"
    frmFrame1.Visible = True
  Case 6:
    frmFrame1.Visible = False
    lblLabel2.Caption = "Dropdown Combo Box"
    cmdAdd.Visible = True
    cboCombo1.AddItem "Radio"
    cboCombo1.AddItem "PC"
    cboCombo1.AddItem "Stereo"
    cboCombo1.AddItem "Television"
    cboCombo1.AddItem "VCR"
    cboCombo1.AddItem "Tape Deck"
    cboCombo1.Visible = True
  Case 7:
    CmdAdd.Visible = False
    cboCombo1.Visible = False
    lblLabel2.Caption = "Listbox Control"
    lstList1.AddItem "Bear"
    lstList1.AddItem "Tiger"
    lstList1.AddItem "Bull"
    lstList1.AddItem "Horse"
    lstList1.AddItem "Rabbit"
    lstList1.AddItem "Dog"
    lstList1.AddItem "Cat"
    lstList1.Visible = True
```

```
    Case Else:
      lblLabel2.Visible = False
      lstList1.Visible = False
End Select
dem = dem + 1
If dem > 8 Then
  dem = 0
End If
End Sub
```

Chọn lệnh View ➡ Object để quay lại cửa sổ Form. Nhấp đúp điều khiển nhãn (Label) để chèn thêm nhãn thứ hai vào mẫu biểu. Đổi thuộc tính Caption của nhãn thành Label control, thuộc tính Alignment của nhãn thành 2–Center, thuộc tính Border style bằng 1–Fixed single, thuộc tính Back Color bằng highlight text, thuộc tính Appearance bằng 0–Flat, thuộc tính Visible bằng False và thuộc tính Name thành lblLabel2.

12. Nhấp đúp điều khiển nhãn (Label) để chèn thêm nhãn thứ ba vào mẫu biểu. Đổi thuộc tính Caption của nhãn thành Text appears in a text control, thuộc tính Visible bằng False, thuộc tính Name bằng lblLabel3.

13. Nhấp đúp điều khiển hộp nhập (biểu tượng ab) trên thanh công cụ để chèn nó vào mẫu biểu. Đổi thuộc tính Text của hộp nhập thành Change this!, thuộc tính Visible bằng False, thuộc tính Name bằng txtText1.

14. Nhấp đúp điều khiển nút lệnh (Command Button) trên thanh công cụ để chèn nút lệnh thứ ba vào mẫu biểu. Thay đổi thuộc tính Caption của nút lệnh thành &Press Me, thuộc tính Visible thành False, thuộc tính Name thành cmdPress, dời nút lệnh tới vị trí thích hợp. Nhấp đúp nút lệnh bật cửa sổ Code và nhập vào những lệnh sau:

```
Private Sub cmdPress_Click()
If cmdPress.Caption = "&Press Me" Then
  cmdPress.Caption = "Once again"
Else
  cmdPress.Caption = "&Press Me"
```

```
End If
Beep
End Sub
```

15. Chọn lệnh View ➥ Object để quay lại cửa sổ Form. Nhấp đúp điều khiển ô chọn (Check Box) trên thanh công cụ để đặt nó vào giữa mẫu biểu. Thay đổi thuộc tính Caption của nó thành Pears, thuộc tính Value thành 1–Checked, thuộc tính Visible bằng False, thuộc tính Name thành chkCheck1.
16. Nhấp đúp điều khiển ô chọn trên thanh công cụ để đặt ô chọn thứ hai vào giữa mẫu biểu. Thay đổi thuộc tính Caption của nó thành Peaches, thuộc tính Visible thành False, thuộc tính Name thành chkCheck2. Di chuyển điều khiển xuống dưới ô chọn Pears.
17. Nhấp đúp điều khiển ô chọn trên thanh công cụ để đặt ô chọn thứ ba vào giữa mẫu biểu. Thay đổi thuộc tính Caption của nó thành Prunes. Đặt thuộc tính Value thành 1–Checked, thuộc tính Visible thành False, thuộc tính Name thành chkCheck3. Di chuyển điều khiển xuống dưới ô chọn Peaches (xem Hình 3.7).
18. Nhấp đúp điều khiển nút tùy chọn (Option Button) trên thanh công cụ để đặt nút tùy chọn đầu tiên vào giữa mẫu biểu. Thay đổi thuộc tính Caption của nó thành Earth, thuộc tính Visible thành False, thuộc tính Name thành optOption1.
19. Tiếp tục lặp lại bước 18 ba lần nữa để tạo nút tùy chọn kế tiếp, với thuộc tính Visible bằng False và thuộc tính Caption của chúng lần lượt là Mars, Mercury, Venus, thuộc tính Name lần lượt là optOption2, optOption3, optOption4 (xem Hình 3.8).
20. Nhấp đúp điều khiển khung (Frame) trên thanh công cụ để đặt một khung vào giữa mẫu biểu. Định lại kích cỡ khung, xóa giá trị Frame1 trong thuộc tính Caption của nó, thuộc tính Visible bằng False, đặt thuộc tính Name bằng frmFrame1.
21. Nhấp điều khiển nút lệnh (Command Button), vẽ nút lệnh trong khung. Thay đổi thuộc tính Caption của nút lệnh thành Frame control. Kế đó, nhấp điều khiển nút tùy chọn, lần lượt vẽ hai nút tùy chọn trong khung (chú ý không nhấp đúp, bạn sẽ hiểu lý do khi đã hiểu rõ về điều khiển khung (Frame). Lần lượt thay đổi thuộc tính Caption của từng điều khiển thành option 1 và option 2 (xem Hình 3.9).

22. Nhấp đúp điều khiển nút lệnh (Command Button) để chèn thêm một nút lệnh vào giữa mẫu biểu. Thay đổi thuộc tính Caption của nó thành &Add, thuộc tính Visible thành False. Thu nhỏ nút lệnh và di chuyển đến vị trí thích hợp (xem Hình 3.10). Nhấp đúp trên nút lệnh Add để mở cửa sổ Code và nhập vào những lệnh sau :

```
Private Sub cmdAdd_Click()
    Combo1.AddItem Combo1.Text
End Sub
```

23. Nhấp đúp điều khiển hộp combo (Combo Box) để chèn hộp combo xổ xuống vào giữa mẫu biểu, đổi thuộc tính Visible của nó thành False, thuộc tính Text của hộp combo bằng rỗng, thuộc tính Name bằng cboCombo1.

24. Nhấp đúp điều khiển hộp danh sách (List Box) để chèn hộp danh sách vào giữa mẫu biểu. Đặt thuộc tính Visible của nó bằng False, thuộc tính Name bằng lstList1. Định kích cỡ và di chuyển hộp danh sách đến vị trí thích hợp (xem Hình 3.11).

25. Mẫu biểu lúc này trong giống như Hình 3.2 với rất nhiều điều khiển chồng lấp lên nhau, hoặc có thể khác do cách di chuyển vị trí các điều khiển của bạn. Bạn đừng lo lắng về điều đó lúc này. Cuối cùng, chọn lệnh File ➡ Save Project để cất toàn bộ ứng dụng của bạn.

26. Chọn lệnh Run ➡ Start hay nhấn phím F5, nếu ứng dụng của bạn hiển thị màn hình đầu tiên tương tự Hình 3.3, xin chúc mừng bạn!

Lưu ý

Trong trường hợp chương trình của bạn không thể thi hành, hoặc vận hành không đúng với những mô tả trong Chương 2 – Bài 3. Bạn đã thực hiện sai điều gì đó trong quá trình thiết kế ứng dụng. Nếu bạn lần đầu thiết kế ứng dụng trong Visual Basic, sai sót là chuyện đương nhiên. Đừng vội nản lòng, cứ bình tĩnh đọc tiếp và quay lại tiếp tục công việc thiết kế của mình khi bạn nghĩ rằng đã hiểu rõ từng điều khiển trong Visual Basic.

Mách nước

Bạn cũng có thể liên lạc tại nhà sách Văn Lang hoặc 145 Nguyễn Đình Chính, P. 11, Q. Phú Nhuận, TP. Hồ Chí Minh để chép đĩa mềm chứa đủ chương trình thực hành minh họa trong tập sách cho tiện việc nghiên cứu. Tuy nhiên, bạn nên cố gắng tự thiết kế tất cả các bài thực hành mô tả trong sách. Đó là cách nhanh nhất nếu bạn muốn trở thành một lập trình viên Visual Basic.

Phụ lục D

Trả lời

CHƯƠNG I

Bài 1

1. Chương trình là một danh sách lệnh báo cho máy tính biết chính xác những gì cần thực hiện.
2. Không có gì tuyệt đối. Máy tính là một cái máy đần độn, không thể hoạt động nếu không có các lệnh chi tiết trong chương trình.
3. Bạn có thể mua một chương trình hay tự viết nó.
4. Chương trình bạn viết thực hiện chính xác những gì bạn muốn một khi bạn đã loại bỏ tất cả lỗi kỹ thuật ra khỏi chương trình.
5. Để viết chương trình, bạn phải cố gắng và mất nhiều thời gian.
6. Đúng
7. Mã lệnh là chương trình.
8. Một lỗi xảy ra tình cờ trong chương trình bạn viết.
9. Lỗi cú pháp và lỗi logic.
10. Bạn đã gặp lỗi cú pháp trong chương trình.
11. Visual Basic sẽ tìm các lỗi cú pháp dùm bạn.
12. Lỗi logic khó tìm hơn lỗi cú pháp.
13. Với các bảng điều khiển phần cứng.
14. Lập trình viên không cần phải là kỹ sư điện tử nữa.
15. On và Off mô tả mức điện thế.
16. Bộ phận bàn phím.
17. Mã lệnh là chương trình mà bạn và các lập trình viên khác tạo.
18. Ngôn ngữ máy
19. FORTRAN
20. BASIC
21. Beginner's All-purpose Symbolic Instruction Code
22. Các ngôn ngữ lập trình thủ tục

23. Các chương trình hướng biến cố
24. Giao diện người dùng dạng đồ họa (Graphical User Interface)
25. Hai biến cố có thể là quá trình nhấn phím hay nhấp mouse.
26. Người dùng có thể kích hoạt biến cố bất kỳ theo thứ tự tùy ý.
27. Sai. Visual Basic yêu cầu một số thủ tục cho quá trình lập trình các hàm xác định.
28. Đúng

Bài 2

1. Đúng
2. Đúng
3. Chọn lệnh File ➥ Run, nhập E:\SETUP. (Bạn có thể nhập lệnh ở dạng chữ thường hay chữ hoa, giả sử là ổ đĩa CDROM.)
4. Giá trị được dùng nếu bạn không nhập một giá trị khác.
5. Chọn File ➥ Exit thoát khỏi Visual Basic và trở lại Windows.
6. Bạn có thể mất một phần hay tất cả chương trình bạn đang viết.
7. Cửa sổ Code, Form, Project, Properties và Toolbox.
8. Visual Basic dùng cửa sổ Form làm nền cho trình ứng dụng.
9. Sai
10. Đúng. Nhấp vào góc trái trên của điều khiển hộp nhập.
11. Dùng phím truy xuất thi hành nhanh hơn lệnh menu.
12. Thanh công cụ sẽ cung cấp nút nhấn truy xuất tới các lệnh menu phổ biến.
13. Thanh menu Visual Basic có cùng chuẩn đặt tên cho phần lớn lệnh menu của nó như các chương trình Windows khác.
14. 11
15. Không phải tất cả các lệnh trên thanh công cụ hay menu luôn có hiệu lực trong mọi thời điểm.
16. Bộ chỉ dẫn đơn vị đo mô tả kích cỡ và vị trí các thành phần trên cửa sổ Form.
17. 1 twip bằng 1/1440 inch.
18. Khung lưới giúp gắn điều khiển vào các vị trí để canh thẳng hàng chúng với các điểm lưới khi bạn thiết đặt lệnh Options Environment thích hợp.

CHƯƠNG II

Bài 3

1. Đầu tiên bạn phải nạp trình ứng dụng bằng cách dùng File ➡ Open.
2. Tập tin project lưu nội dung mô tả của trình ứng dụng.
3. .VBP
4. Nạp project và nhìn vào cửa sổ Project.
5. Cho phép người dùng tương tác với chương trình.
6. Cửa sổ Toolbox lưu điều khiển bạn có thể thêm vào mẫu biểu.
7. Người dùng cần thoát chương trình một cách dễ dàng.
8. Sai. Người dùng không thể thay đổi trực tiếp văn bản được hiển thị trong điều khiển nhãn.
9. Điều khiển hộp nhập (Text Box).
10. Bạn có thể điều khiển kích cỡ phông chữ và kiểu văn bản.
11. Người dùng có thể di chuyển con trỏ văn bản tới trước và sau một hộp nhập cũng như sử dụng các phím Ins và Del để chèn và xóa văn bản.
12. Điều khiển nút lệnh (Command Button).
13. Sai
14. Đúng
15. Người dùng bỏ chọn một ô chọn bằng cách lựa ô đã chọn.
16. Điều khiển khung (Frame).
17. Đúng
18. Đúng
19. Sai
20. Đúng
21. Mẫu biểu làm việc như nền trình ứng dụng và lưu điều khiển bạn đặt trên nó.
22. Đúng
23. Người dùng nhấp nút lệnh bằng mouse hay nhấn Alt+phím truy xuất nhanh nếu nhãn của nút lệnh có ký tự gạch dưới. Cách thứ

ba khá hay (không được đề cập trong bài này): Người dùng có thể nhấn Tab cho đến khi nút lệnh hiện sáng và nhấn Enter.

24. Nút lệnh cạnh hộp combo báo cho Visual Basic biết rằng dữ liệu được bổ sung vào danh sách.
25. An Huy nên thay thế hộp combo đơn giản thành điều khiển hộp combo xổ xuống.

Bài 4

1. Một biến cố có thể là phím nhấn, di chuyển mouse, nhấp mouse, chọn menu, hay bất kỳ điều gì khác mà người dùng có thể thực hiện để đáp ứng chương trình.
2. Có quá nhiều cách cho người dùng tương tác với các điều khiển có thể nhìn thấy trên màn hình. Chương trình dựa trên nền văn bản thi hành lệnh theo thứ tự định sẵn của người dùng.
3. Sai. Đôi khi Windows chặn các biến cố cho hệ thống của riêng nó sử dụng.
4. Giúp phân biệt hoạt động và hành vi của điều khiển.
5. Thủ tục biến cố
6. Sau đây là 4 thuộc tính (từ danh sách thuộc tính khổng lồ): font size, font style, size, và color.
7. Đúng
8. Đúng
9. Cửa sổ Properties
10. Quá trình cập nhật chương trình sau đó.
11. Ba ký tự tiền tố mô tả loại điều khiển nào bạn đang làm việc.
12. Người dùng sẽ thích ứng nhanh với chương trình của bạn khi chúng tuân theo các chuẩn Windows.
13. AUTOLOAD.VBP
14. Đúng.
15. Tên nên được gán với các giá trị dễ nhớ.
16. Phần đầu tiên của thủ tục biến cố là tên điều khiển, tiếp sau là ký tự _, sau đó là tên biến cố (và cặp dấu ngoặc đơn để bạn biết bạn đang làm việc với tên thủ tục, không phải tên điều khiển)

17. Mã đầu cuối
18. End
19. Các biến cố của điều khiển và mẫu biểu trong project.
20. Thông thường project và mẫu biểu được đặt cùng tên với phần mở rộng là .VBP và .FRM.
21. Mẫu biểu
22. Hộp combo
23. Nút lệnh
24. An Huy à, ít phông chữ vẫn tốt hơn bằng không bạn sẽ làm người dùng bối rối khi nhìn màn hình của bạn.
25. A. End là một từ khóa.

    B. Tên không thể bắt đầu bằng một số.

    C. Tên không thể đặc biệt như là dấu $.

    D. Tên không thể có ký tự gạch nối -.

CHƯƠNG III

Bài 5

1. Điều khiển hiện có sự chú ý của Windows là điều khiển đang được hiện sáng.
2. Thứ tự mà các điều khiển sẽ nhận tiêu điểm khi người dùng nhấn phím Tab.
3. TabIndex
4. Thuộc tính Name.
5. Sai
6. Đúng
7. Icon là một hình.
8. Point bằng 1/72 inch.
9. Là nơi bạn có thể chọn màu.
10. D. Tất cả đều đúng.
11. Enabled.
12. 0, 1, hay 2 tương ứng với canh trái, giữa hoặc phải.

13. Sai; hiếm khi bạn phải ấn định hay thay đổi nhiều thuộc tính bởi vì giá trị mặc định là phổ biến cho nhiều thuộc tính.
14. Ký tự trở lại đầu dòng gửi con trỏ văn bản tới dòng kế tiếp trên màn hình.
15. Quá trình trao đổi dữ liệu động (Dynamic Data Exchange).
16. Đúng tùy theo thuộc tính Visible.
17. Left, Right, Top, Width.
18. Điều khiển nhãn (Label).
19. Thuộc tính PasswordChar.
20. Thuộc tính Cancel.
21. Dùng điều khiển hộp combo xổ xuống.

CHƯƠNG III

Bài 6

1. Đúng, mẫu biểu là một đối tượng như những điều khiển khác.
2. Sai, điều khiển nhãn không có thuộc tính MultiLine.
3. Pixel là thành phần màn hình nhỏ nhất có hiệu lực.
4. Thuộc tính WindowState.
5. Cắt bớt một phần văn bản.
6. Sai
7. Sai, điều khiển hộp nhập có các thanh cuốn.
8. Định thuộc tính AutoSize là False.
9. Sai.
10. Sai.
11. Thuộc tính TabIndex điều khiển thứ tự tiêu điểm.
12. Biến cố Load.
13. Biến cố Load xảy ra trước biến cố Activate.
14. Cửa sổ Code có hộp danh sách thủ tục xổ xuống chứa tất cả biến cố của đối tượng bất kỳ.
15. Nhãn không bao giờ nhận tiêu điểm (focus).

16. Tên thủ tục biến cố phải là txtLastName_Change().
17. Thuộc tính KeyPreview xác định liệu mẫu biểu hay điều khiển nhận phím nhấn.
18. Thuộc tính Caption.
19. Bổ sung phím truy xuất vào nhãn mô tả nội dung hộp nhập. Hãy chắc chắn rằng thuộc tính TabIndex của nhãn nhỏ hơn 1 so với giá trị TabIndex của hộp nhập.
20. Nhãn có thể giấu nhiều đối tượng mẫu biểu khác.
21. Nhãn sẽ tự mở rộng theo chiều ngang trước khi bạn có cơ hội định thuộc tính WordWrap.

CHƯƠNG IV

Bài 7

1. Vị trí lưu trữ được đặt tên trong bộ nhớ.
2. Lệnh Dim
3. Kiểu dữ liệu là một tập hợp dữ liệu có tất cả giá trị Visual Basic nằm trong đó.
4. Integer, Long, Single, Double, Currency, Variant, String
5. Integer
6. Double
7. Single, Double, và Currency
8. E nghĩa là lũy thừa
9. Ký hiệu khoa học là ký hiệu viết tắt các giá trị số lớn.
10. Sai
11. Đúng.
12. Bạn sẽ giúp ngăn chặn các lỗi kỹ thuật có thể xảy ra khi đánh vần sai tên biến.
13. Chuỗi có độ dài cố định chỉ có thể nhận một số ký tự định trước trong khi chuỗi có độ dài thay đổi có thể nhận các chuỗi có chiều dài thay đổi khi chương trình thi hành.
14. Đúng
15. Đúng

16. A. 7

    B. 9

    C. 16

    D. (cần bao quanh với cặp dấu ngoặc đơn để tính giá trị trung bình đúng)

    E. 6

    F. 40

17. Toán tử là từ hay ký hiệu thực hiện phép tính và xử lý dữ liệu.
18. Toán tử + được dùng cho cả phép cộng lẫn nối chuỗi (tùy theo lúc nó sử dụng) và toán tử & chỉ dùng để nối chuỗi.
19. Dim SqFoot As Single
20. SalesPrice = Price / Discount

    Tax = TaxRate * SalesPrice
21. Name là một từ khóa và không thể là tên biến.
22. An Huy không thể định nghĩa nhiều hơn một biến cùng tên.
23. An Huy không thể dùng dấu $ hay dấu, trong các giá trị hằng.

Bài 8

1. Điều kiện là quan hệ có thể kiểm tra trong chương trình
2. Sai, biểu thức kiểm tra điều kiện chỉ đưa ra một trong hai kết quả, đúng hoặc sai
3. Sai nhưng cặp ngoặc đơn có thể phân biệt mã lệnh
4. <, >, <=, >=, =, <>
5. A. Sai

   B. Đúng

   C. Đúng

   D. Đúng

   E. Đúng

   F. Sai
6. Bảng ASCII
7. Lệnh If giúp đưa ra quyết định (cũng như lệnh Select Case)
8. Lệnh If-Else

9. Sai
10. Lệnh If lồng nhau là lệnh If không có lệnh If khác bên trong
11. Lệnh Select Case
12. 4: Case Value, Case Is, Case To, và Case Else
13. Không có gì xảy ra và điều khiển bắt đầu lệnh ngay sau lệnh End Select.
14. Mã lệnh trong Case Else sẽ thi hành trước khi trả lại điều khiển cho lệnh ngay sau lệnh End Select.
15. Case To
16. Case Is
17.
```
If (M = 3) AND (P = 4) Then
    TestIt = "Yes"
    End If
```
18.
```
If (d >= 3) Or (p < 9) Then
```
19.
```
Select Case Age
    Case Is <0: lblTitle.Text = "Age Error"
    Case Is <5: lblTitle.Text = "Too young"
    Case 5: lblTitle.Text = "Kindergarten"
    Case 6 To 11: lblTitle.Text = "Elementary"
    lblSchool.Text = "Lincoln"
    Case 12 To 15: lblTitle.Text = "Intermediate"
    lblSchool.Text = "Washington"
    Case 16 To 18: lblTitle.Text = "High School"
    lblSchool.Text = "Betsy Ross"
    Case Else: lblTitle.Text = "College"
    lblSchool.Text = "University"
    End Select
```
20. Không có lệnh End Else.

CHƯƠNG V

Bài 9

1. Ghi chú giúp giải thích logic chương trình.

2. Để giúp lập trình viên trong quá trình bảo trì chương trình.
3. Sai
4. Đúng
5. 2: lệnh Rem và dấu lược (').
6. Để nói rõ tên của lập trình viên và ngày viết chương trình. Để mô tả trong thủ tục (General) mục đích tổng quát của chương trình.
   Để mô tả ở đầu mỗi thủ tục mục đích tổng quát của thủ tục đó. Để giải thích các lệnh khó khăn và phức tạp giúp lập trình viên sửa đổi chương trình, sau đó có thể hiểu các dòng lệnh mà không phải tìm hiểu những mã lệnh khó hiểu.
7. Lập trình viên
8. Sai; thêm chú thích khi bạn lập trình để có thể hiểu chương trình dễ dàng hơn khi bạn sửa đổi nó sau này.
9. Sai
10. Xác định giá trị nhập cho biết nhóm nút lệnh nào bạn cần.
11. Bằng cách xác định giá trị msg.
12. Modal cho biết liệu người dùng phải đáp ứng một hộp thông báo hay không trước khi chuyển sang trình ứng dụng khác.
13. Đúng
14. Sai
15. Để nhận câu trả lời
16. Chuỗi và Variant
17. Kiểm tra chuỗi rỗng, "", để xem người dùng đã nhấn Cancel hay OK.
18. Nhấp nút lệnh OK (hay nhấn Enter).
19. 4
20. Không
21. Dùng giá trị nhập thích hợp
22. 7
23. 7

24. Chắc chắn nhập đối số có giá trị bằng 4096 hay tên hằng MB_SYSTEMMODAL.
25. An Huy chỉ có thể dùng dấu ' nếu muốn thêm chú thích ở cuối một dòng. An Huy phải thay đổi mã lệnh thành:

```
do until (endOfFile) ' Continue until the end
```

Bài 10

1. Vòng lặp là quá trình lặp lại một khối lệnh Visual Basic.
2. 5
3. 4
4. Đúng
5. Vòng lặp vô tận (ít nhất cho đến khi người dùng can thiệp để dừng vòng lặp)
6. Để duy trì quá trình đợi người dùng trả lời, dùng một vòng lặp, cho đến khi người dùng nhập vào câu trả lời đúng.
7. 9
8. 10
9. 0
10. 10
11. 1
12. Lệnh For
13. Lệnh Do
14. Vòng lặp Do While tiếp tục thi hành trong khi biểu thức kiểm tra quan hệ là true (đúng), ngược lại vòng lặp Do Until tiếp tục thi hành trong khi biểu thức kiểm tra quan hệ là false (sai).
15. Visual Basic sẽ kiểm tra biểu thức kiểm tra quan hệ ở đầu vòng lặp bằng cách dùng lệnh Do While, ngược lại Visual Basic sẽ kiểm tra biểu thức kiểm tra quan hệ ở cuối vòng lặp bằng cách dùng lệnh Do-Loop While.
16. Quá trình nhắc đi nhắc lại của một vòng lặp.
17. Dùng giá trị Step âm.
18. Giá trị Step của số gia phải âm.
19. Lệnh Exit Do và Exit For kết thúc vòng lặp trước quá trình dừng tự nhiên của chúng.

20. Dim Guess, Ans As Integer

```
Guess = 34 ' The user must guess this
Do Ans = InputBox("Make a guess...", "Guessing Game")
' Tell the user about the wrong guess
If (Ans <> Guess) Then
MsgBox "You guessed incorrectly. Try again."
End If
Loop While Ans <> Guess
```

21. An Huy thay đổi biến i trong vòng lặp, luôn duy trì quá trình kích hoạt biến điều khiển vòng lặp For ở cuối vòng lặp.
22. An Huy phải đổi giá trị bắt đầu và kết thúc như sau :

```
For I = 1 To 100 Step 1
```

CHƯƠNG VI

Bài 11

1. Mảng giúp bạn duyệt qua số lượng dữ liệu lớn mà không cần tham khảo nhiều tên biến.
2. Một mục đơn từ danh sách các giá trị của mảng.
3. Chỉ số mảng là số tham khảo đến các thành phần mảng khác nhau.
4. 1
5. 20
6. Đúng
7. Sử dụng chỉ số
8. Lệnh Static định nghĩa mảng
9. 1
10. Vòng lặp For làm việc tốt nhất cho quá trình xử lý mảng.
11. Hộp danh sách và hộp combo
12. Sai
13. Đúng khi ấn định thuộc tính MultiSelect là True.
14. Đúng

15. Phương thức khởi tạo, đếm và gỡ bỏ các mục khỏi điều khiển hộp danh sách và hộp combo.
16. Phương thức AddItem
17. Đúng
18. Sai
19. Đúng
20. Đúng
21. Phương thức Selected
22. Sai
23. Thuộc tính Sorted
24. Static StdAges(3) As Integer
25. Thuộc tính MultiSelect
26.
```
Sub cmdGoAway_Click()
lstVals.Clear
End Sub
```
27. Nút lệnh có thể kích hoạt thủ tục biến cố sử dụng phương thức AddItem để thêm một mục mới vào hộp combo.
28. An Huy không thể định nghĩa hộp danh sách nếu hộp danh sách là một mảng biến. Carla phải dùng điều khiển hộp danh sách trên cửa sổ Toolbox để đặt hộp danh sách lên mẫu biểu.

Bài 12

1. Điều khiển nút tùy chọn (Option Button)
2. Điều khiển ô chọn (Check Box)
3. Nút tùy chọn mô phỏng các nút đài cũ với bộ chọn đài bằng cách nhấn nút.
4. Ban đầu nút tùy chọn sẽ không được chọn.
5. Đúng
6. Thuộc tính Alignment xác định nội dung mô tả nút tùy chọn hay ô chọn sẽ xuất hiện ở bên phải hay trái nút hay ô chọn.
7. Thuộc tính Value
8. Thuộc tính Value

9. Sai
10. Đúng
11. Một bộ lưu trữ các điều khiển khác như là nút tùy chọn.
12. Sai
13. Ký tự đặc biệt thực hiện một hành động trên màn hình.
14. Nối tổ hợp chuỗi ký tự trở lại đầu dòng / xuống dòng trong văn bản để chia dòng thành từng dòng nhãn riêng biệt.
15. Đổi số thành ký tự ASCII tương ứng.
16. Mảng điều khiển cùng kiểu điều khiển.
17. Tất cả mảng, kể cả mảng điều khiển, chỉ có một tên.
18. Bằng giá trị chỉ số
19. 0
20. 1
21. Index
22. Bằng cách hiển thị hộp thông báo bảo đảm bạn muốn tạo mảng điều khiển.
23. cmdAll_DblClick (Index As Integer)
24. An Huy cần vẽ các nút tùy chọn trong khung, mà không di chuyển chúng vào khung.

CHƯƠNG VII

Bài 13

1. Đối số là giá trị chuyển tới hàm trong cặp dấu ngoặc đơn của nó.
2. Sai
3. Đúng
4. Sai
5. Có 3 hàm số nguyên khác nhau bởi vì có nhiều cách khác nhau để làm tròn số.
6. A. 61

   B. -62

C. –61

D. 422

7. Sai; CInt() là hàm làm tròn trong khi Int() và Fix() luôn làm tròn các đối số dương thành số nguyên nhỏ hơn hay bằng đối số.
8. Hàm Asc() là ảnh phản chiếu của hàm Chr$().
9. A. "ABCDEFG"

   B. "abcdefg"
10. Bất kỳ điều gì lưu trong chuỗi "78.1"
11. A. S

    B. Sam

    C.

    D. ams

    E. ams
12. Đúng, xem câu hỏi #18 và #19.
13. Đúng
14. Str$() (hay Str())
15. Chuyển giá trị số đến Str$() trước khi nối giá trị tới chuỗi dấu nhắc.
16. Gửi tới hàm Len() một biến chính xác kép và nhân kết quả với 250.
17. Dim ANum, ANumSq As Single

    ANum = Val(InputBox$("Enter a number to square"))

    MsgBox "The square of your number is" & Str$(Sqr(ANum))
18. First = InputBox$("What is your first name?")

    Last = InputBox$("What is your last name?")

    MsgBox "Your name has" & Str$(Len(First) + Len(Last)) & "letters"
19. ValAsc = Asc("P")
20. An Huy cần đổi giá trị trọng lượng thành một số nguyên dài bởi vì anh ấy đang tính một giá trị cho kết quả số nguyên quá lớn.

Bài 14

1. Đúng
2. Hàm ngày giờ lấy giá trị từ đồng hồ và lịch bên trong máy tính.
3. Thời gian 24 giờ không được dùng với chỉ báo AM hay PM. Hãy cộng 12 cho tất cả thời gian sau 12:59 PM để có được thời gian 24 giờ.

4. Đúng
5. Sai
6. 19:54
7. Date$() trả lại chuỗi và Date() trả lại kiểu dữ liệu Variant.
8. Sai; Visual Basic chỉ hỗ trợ hàm Now().
9. Now() trả về giá trị dựa trên đồng hồ 12 giờ.
10. Sai
11. Sai; Date và Time (là các lệnh không phải hàm) định ngày giờ cho máy tính.
12. Date$()
13. Timer() trả về số giây từ giữa đêm với giá trị số chính xác kép.
14. Byte là một ký tự trong bộ nhớ.
15. Kiểu dữ liệu VarType 7
16. TimeValue() nhận hai giá trị đối số, giờ, phút, giây, và tạo thành giá trị thời gian trong khi hàm Hour(), Minute(), và Second() nhận giá trị thời gian đầy đủ và trả lại từng thành phần thời gian riêng biệt.
17. Giá trị logic là kết quả của phép so sánh quan hệ.
18. Format() và Format$()
19. Format() trả về kiểu dữ liệu Variant và Format$() trả về một chuỗi.
20. Dấu phân cách hàng ngàn là dấu phẩy (hoặc chấm cho một số thiết đặt quốc tế) sẽ phân cách từng nhóm 3 số tính từ bên trái dấu thập phân trong giá trị lớn hơn 999.
21. Chuỗi định dạng cố định "Fixed" không bao gồm giá trị âm trong cặp ngoặc đơn và không hiển thị dấu phân cách hàng ngàn trong các số lớn.
22. Time$ không chấp nhận giá trị 12-giờ chứa chỉ báo AM hay PM. Bạn có thể sử dụng hàm Right$() để biết người dùng đã nhập vào ký tự AM hoặc PM hay chưa.

CHƯƠNG VIII

Bài 15

15. Về mặt kỹ thuật, câu trả lời là sai. Tuy nhiên, chúng ta không chẻ nó làm tư. Thủ tục biến cố là dạng đặc biệt của thủ tục Subroutine.
16. Dùng lệnh Call để gọi thủ tục Subroutine.
17. Dùng tên thủ tục Function trong biểu thức hay lệnh.
18. Bạn có thể chọn lệnh từ thanh menu View hay nhập Sub hoặc Function trong thủ tục khác tại cửa sổ Code.
19. Nếu dấu $ xuất hiện sau tên hàm, hàm sẽ trả về một chuỗi.
20. Thủ tục Function trả về một giá trị cho thủ tục gọi nó.
21. Gán giá trị trả về của hàm cho tên hàm, xem như tên hàm là một biến.
22. F2
23. Lập trình viên phải cung cấp mã lệnh cho thủ tục Function.
24. Module
25. .BAS
26. Option Base và Option Explicit
27. Đúng
28. Đúng

15. Sub PrReport()

    End

16. Function GetValue()

    End Function

17. GetPi = 3.14159
18. Không có cặp dấu ngoặc đơn quanh danh sách đối số cho nên bạn không thể dùng từ khóa Call.

Bài 16

1. Cục bộ
2. Module

3. Toàn cục
4. Sai
5. Đúng
6. Đúng
7. Toàn cục
8. Module
9. Hằng toàn cục
10. Định nghĩa tên các giá trị hằng
11. Không nơi nào khác trong chương trình bạn được phép gán các giá trị cho hằng.
12. Trong thủ tục (General) của module không mẫu biểu.
13. Đúng; một biến có thể cục bộ trong một thủ tục còn biến kia thì cục bộ trong thủ tục khác. Biến cục bộ cũng có thể cùng tên với biến toàn cục. Biến cục bộ sẽ là biến hoạt động trong thủ tục của nó.
14. Giá trị trong biến cục bộ luôn mất đi khi thủ tục kết thúc.
15. Thủ tục đang gửi
16. Thủ tục đang nhận
17. Sai
18. Biến chuyển tới hàm gọi là biến cục bộ cho nên hàm nhận cần biết kiểu dữ liệu của đối số đã nhận.
19. Sai
20. Sai; bạn có thể chuyển nhiều đối số đến một thủ tục Function nhưng nó chỉ trả về một giá trị.
21. 2
22. Dữ liệu được chuyển có thể thay đổi bởi thủ tục nhận và những thay đổi này vẫn giữ nguyên hiệu lực khi thủ tục gửi chiếm lại quyền điều khiển.
23. Dữ liệu được chuyển không thể thay đổi bởi thủ tục nhận ngoại trừ thay đổi chỉ sử dụng trong phạm vi thủ tục nhận.

24. Sử dụng từ khóa ByVal hay bao các đối số được chuyển trong cặp dấu ngoặc đơn.

25. Hàm UBound()

26. Global LastName As String

    Global Age As Integer

27. Global Const NDAYS = 7

    Global Const NMONTHS = 12

28. A. Bạn phải khởi tạo một hằng và không thể khai báo kiểu dữ liệu cho hằng.

    B. Không thể khai báo kiểu dữ liệu cho hằng.

    C. Luôn định nghĩa hằng bằng cách dùng từ khóa Const.

    D. Bạn không thể khởi tạo biến toàn cục tại thời điểm bạn định nghĩa biến.

29. An Huy không nên viết giá trị chỉ số trong cặp dấu ngoặc của mảng nhận. Hãy xóa chỉ số trong cặp ngoặc đơn và dùng hàm UBound() trong thủ tục để tìm chỉ số cao nhất của mảng.

CHƯƠNG IX

Bài 17

1. Dữ liệu trong tập tin

2. Người dùng có thể chọn từ ổ đĩa, đường dẫn, và tên tập tin khi dùng hộp thoại. Người dùng có thể rất dễ gây ra lỗi khi nhập tên tập tin vào hộp nhập.

3. Tập hợp những điều khiển nhận thông tin người dùng.

4. Thêm khung vào mẫu biểu để lưu các điều khiển tập tin. Khung cho bạn nhóm các điều khiển tập tin lại với nhau mà không gây trở ngại cho các điều khiển khác như hộp danh sách chẳng hạn.

5. Định thuộc tính Visible của khung bằng True hoặc False.

6. 3

7. Trong thời gian thi hành chương trình.

8. Một mẫu, với các ký tự trong tên tập tin là * và ? sẽ xác định nhóm những tập tin được hiển thị trong hộp danh sách tập tin.

9. Khi một điều khiển tập tin thay đổi, điều đó sẽ ảnh hưởng đến các điều khiển tập tin khác.
10. Quản lý hộp danh sách ổ đĩa đầu tiên bởi vì thay đổi trong hộp danh sách ổ đĩa sẽ ảnh hưởng đến các điều khiển tập tin khác.
11. Điều khiển hộp danh sách tên tập tin và thư mục.
12. Điều khiển hộp danh sách tên tập tin là điều khiển duy nhất cần được cập nhật khi người dùng thay đổi hộp danh sách thư mục.
13. Biến DidCancel có giá trị True hoặc False.
14. Biến cố Change.
15. Đúng
16. Cả hai ACCT* và ACCT*.* đều làm việc.
17. Cả hai *.ex? và *.ex* đều làm việc.
18. drvDisk.Drive = "d:"
    dirList.Path = "d:\vbprimer\direng" ' Root directory

Bài 18

1. Tập hợp dữ liệu, chương trình hay tài liệu liên quan.
2. Lưu dữ liệu dài.
3. Đúng khi các tập tin được lưu trong những thư mục hay đĩa riêng biệt.
4. Open
5. Input, Output, và Append
6. Thẻ File kết hợp tập tin với một giá trị số.
7. Visual Basic tạo tập tin.
8. Visual Basic ghi đè tập tin.
9. Visual Basic tạo tập tin.
10. Visual Basic thêm vào cuối tập tin.
11. Visual Basic hiển thị lỗi.
12. 1 đến 255
13. Sai
14. FILES=
15. FreeFile()

16. Close
17. Close đảm bảo tập tin được lưu an toàn trong trường hợp gặp sự cố về điện.
18. Lệnh sẽ đóng tất cả tập tin đang mở.
19. Write#
20. Dấu -
21. Dấu $
22. Dấu ?
23. Input#
24. Một dòng từ tập tin
25. Hàm Eof() kiểm tra kết thúc tập tin.
26. Line Input#
27. "Peach", 34.54, 1, "98"
28. An Huy phải dùng lệnh Line Input# sau:

```
Line Input #1, MyRecord
```

29. Close 3, 19
30. Open "names.dat" for Append As #7
31. Sub ReadData ()

```
' Reads array data from a file and reports the data
' Assume that 200 values were read
Static CNames(200) As String, CBalc(200) As Currency
Static CDate(200) As Variant, CRegion(200) As Integer
Dim NumVals As Integer ' Count of records
Dim ctr As Integer ' For loop control
NumVals = 1 ' Start the count
' Reads the file records assuming
' four values on each line
Open "c:\mktg.dat" For Input As #1
Input #1, CNames(NumVals), CBalc(NumVals), CDate(NumVals),
CRegion(NumVals)
Do Until (Eof(1) = True)
NumVals = NumVals + 1 ' Increment counter
```

```
If (NumVals = 201) Then
MsgBox "First 200 values are now read", MB_ICONEXCLAMATION
Exit Do
End If
Input #1, CNames(NumVals), CBalc(NumVals), CDate(NumVals),
CRegion(NumVals)
Loop
' When loop ends, NumVals holds one too many
NumVals = NumVals – 1
' The following loop is for reporting the data
For ctr = 1 To NumVals
' Code goes here that outputs the array
' data to the printer
' Next ctr
Close #1
End Sub
```

CHƯƠNG X

Bài 19

1. Cửa sổ Menu Design thiết kế menu.
2. Đúng
3. Sai
4. Đúng
5. mnu
6. Menu xổ xuống sẽ xuất hiện khi bạn nhấp một mục trên thanh menu.
7. Nút lệnh mũi tên lên và xuống sẽ giúp bạn sắp xếp lại các tùy chọn menu.
8. Nút lệnh mũi tên trái và phải giúp bạn thụt dống hay gỡ bỏ dòng xác định các mục menu con.
9. Checked
10. Visible
11. Đúng

12. Đúng
13. Click
14. Chr$(8) (ký tự backspace) sẽ canh phải các mục trên thanh menu.
15. Đúng
16. An Huy cần nối ký tự Chr$(8) trong lúc thi hành thủ tục Form_Load().
17. mnuWindowSplit
18. mnuViewBar1 và mnuViewBar2
19. Chọn thuộc tính Enabled để làm xám mục menu xác định

Bài 20

1. Điều khiển bộ định thời (Timer)
2. Thuộc tính Interval
3. Người dùng không thấy điều khiển bộ định thời (Timer) trên mẫu biểu lúc thi hành chương trình.
4. Không có gì.
5. Sai; điều khiển bộ định thời (Timer) sẽ kích hoạt biến cố riêng của nó.
6. Mili giây
7. Sai
8. 1/1000 giây
9. 10 giây
10. 1 phút
11. tmr
12. Thuộc tính Interval có thể lưu các giá trị lớn hơn số nguyên có thể lưu.
13. Timer()
14. Sai; bạn chỉ có thể kích hoạt biến cố Timer để diễn ra quá trình lặp
15. Môi trường Windows đôi khi có thể làm cho trình ứng dụng Visual Basic nhảy qua một khoảng thời gian.
16. Thêm điều khiển bộ định thời (Timer) vào project với giá trị thuộc tính Interval bằng 1000 để biến cố Timer diễn ra mỗi giây. Cập

nhật thời gian cần hiển thị trong thủ tục biến cố Timer() của điều khiển bộ định thời.

CHƯƠNG X

Bài 21

1. Print
2. Trình quản lý in trong Windows tập hợp tất cả dữ liệu cần in và gửi ra máy in riêng.
3. Trình quản lý in trong Windows chấp nhận tất cả máy in được khởi tạo để chương trình không phải bận tâm.
4. Online (trực tuyến) nghĩa là máy in đã sẵn sàng và có giấy.
5. Kết quả hộp thông báo là cảnh báo người dùng những gì sẽ xuất ra kế tiếp.
6. Đối tượng Printer là đối tượng chung trong Visual Basic, nó tập hợp tất cả dữ liệu in ra và gửi tới trình quản lý in trong Windows.
7. Đúng
8. Điểm nhỏ nhất trên máy in hay màn hình
9. Phương thức Print
10. Sai
11. Sai
12. Mỗi số dương có một dấu cộng ẩn không được in.
13. Một vùng in là kiểu điểm tab có sẵn chiếm 14 cột dữ liệu cần in.
14. Đúng
15. Dùng Chr$(34)
16. Đúng
17. An Huy không tính toán đến thực tế là máy in trong Windows hỗ trợ nhiều phông chữ với những kích cỡ khác nhau đòi hỏi cần tính toán để tìm chiều dài trang thích hợp.
18. Lệnh Print thứ hai sẽ chèn 10 khoảng trống trước khi in, trong khi lệnh in đầu tiên sẽ bắt đầu in tại cột thứ 10.
19. Line 1

20. Line 2
21. The Spanish N is đ.
22. Printer.Print Tab(57); "America"

Bài 22

1. Các điều khiển đường kẻ (Line) và hình dạng (Shape) giúp bạn vẽ các dạng hình học trên mẫu biểu.
2. 6
3. BorderStyle
4. Thuộc tính Shape
5. Điều khiển Picture Box và điều khiển Image
6. Điều khiển Image hiệu quả hơn điều khiển Picture Box.
7. Bitmap, Metafiles, và Icons.
8. Thuộc tính Shrink
9. Thuộc tính Picture
10. LoadPicture()
11. An Huy không thể thay đổi dáng vẻ đường kẻ với các đường dày hơn 1 twip.
12. An Huy phải vẽ nhiều đường mảnh hơn để tạo dáng đường gạch gạch đơn.
13. Không có gì xuất hiện khi hàm LoadPicture() thi hành vì đối số trống.
14. Xác định viền màu xanh da trời bằng cách dùng thuộc tính BorderColor, đường chéo màu đỏ bằng cách dùng thuộc tính FillColor, phía bên trong màu xanh lá cây bằng cách dùng thuộc tính BackColor.

CHƯƠNG XII

Bài 23

1. 2
2. Người dùng có thể chọn vị trí tương đối trên thanh cuốn thay vì nhập các giá trị xác định.

3. Số trên thanh cuốn có thể thay đổi khi người dùng điều chỉnh giá trị thanh cuốn.
4. Bằng cách nhấp trong hoặc trên thanh cuốn.
5. Giá trị nhỏ nhất và lớn nhất mà thanh cuốn có thể hiển thị.
6. hsb và vsb được xem là các tiền tố tên
7. Đúng
8. Chắc chắn tập tin chuyên biệt của khung lưới được thêm vào project.
9. GRID.VBP
10. Sai
11. Vùng giao của một hàng và một cột trong khung lưới.
12. Sai
13. Khi kích cỡ khung lưới không đủ lớn để hiển thị khung lưới đầy đủ.
14. Row và Col
15. SelEndCol, SelEndRow, SelStartCol, và SelEndCol
16. 12
17. MousePointer
18. Caret
19. Cursor
20. Khi thuộc tính MousePointer mặc định của điều khiển không phải là mũi tên. Điều này thường xảy ra chỉ với các điều khiển chuyên biệt.
21. Đồng hồ cát
22. Qua các đối số thủ tục biến cố
23. Đúng
24. Số hàng và cột cố định phải nhỏ hơn tổng số hàng và cột, tổng số hàng và cột phải không nhỏ hơn 2. Bạn sẽ phải thay đổi thuộc tính Enabled của khung lưới thành False nếu bạn không muốn người dùng có bất kỳ điều khiển nào trên đường lưới.
25. Min: 32, Max: 212, SmallChange: 3, LargeChange: 8.

Bài 24

1. Lỗi logic và cú pháp
2. Lỗi cú pháp
3. Trình gỡ rối là công cụ trực tuyến giúp bạn dò tìm lỗi kỹ thuật trong chương trình.
4. Dùng lệnh Options Environment để đáp ứng quá trình kiểm tra cú pháp tác động lẫn nhau.
5. Đúng
6. Sai
7. Lỗi cú pháp dễ tìm hơn lỗi logic.
8. Chế độ thiết kế, chế độ thi hành và chế độ dừng
9. Chế độ thiết kế
10. Nhìn thanh tiêu đề của Visual Basic để biết tên chế độ hiện hành.
11. Chế độ dừng
12. Nhấn Ctrl+Break, chọn lệnh từ menu Run, nhấp nút dừng trên thanh công cụ, hoặc đặt một điểm dừng
13. Một dòng xác định mà Visual Basic dừng quá trình thi hành trong khi chương trình đang chạy.
14. Bằng cách hiện sáng dòng lệnh và nhấn F9, nhấp trên thanh công cụ, hay chọn từ menu Debug.
15. Visual Basic hiện sáng dòng lệnh có điểm dừng
16. Khi gặp một điểm dừng, Visual Basic sẽ dừng chương trình đang chạy trước khi dòng lệnh có điểm dừng sẽ thi hành.
17. Quá trình thi hành chương trình từng dòng một, bạn có thể tự điều khiển thay vì Visual Basic.
18. Nút single footprint yêu cầu Visual Basic thực hiện từng dòng lệnh và double footprint yêu cầu Visual Basic thực hiện từng lệnh trong thủ tục hiện hành tại điểm dừng nhưng không thực hiện từng bước trong các thủ tục được gọi bởi thủ tục hiện hành.
19. Từ hộp thoại Instant Watch, bạn có thể thấy các giá trị biến và điều khiển. Từ hộp thoại Add Watch, bạn có thể yêu cầu Visual Basic chuyển sang chế độ dừng khi một biểu thức xác định trở thành True hay khi một giá trị xác định thay đổi.

20. Instant Watch là công cụ thuận tiện nhất cho quá trình tìm biến tại một điểm dừng.
21. Cửa sổ Debug
22. Phương thức Print
23. Câu lệnh gán
24. An Huy phải đặt đối tượng Debug trước phương thức Print.

Mục lục

CHƯƠNG V

CHƯƠNG VI

CHƯƠNG VII

CHƯƠNG VIII

CHƯƠNG IX

CHƯƠNG X

CHƯƠNG XI

CHƯƠNG XII



GIÁO TRÌNH HỌC VÀ THỰC HÀN[H]

Microsoft Visual Basic căn bản

Giáo trình học và thực hành Visual Basic căn bản là tập hợp những rút tỉa từ kinh nghiệm làm việc thực tế và những bài giảng dễ hiểu nhất của tác giả trong quá trình lập trình và giảng dạy..

Giáo trình học và thực hành Visual Basic căn bản gồm 24 bài học, cuối mỗi bài đều có câu hỏi ôn tập và bài tập thực hành, hướng dẫn từng bước một, qua từng thao tác và giao diện của chương trình, cho phép dễ dàng và nhanh chóng nắm bắt ngôn ngữ lập trình Visual Basic một cách nhanh nhất.

Giáo trình học và thực hành Visual Basic căn bản dành cho mọi đối tượng từ học sinh trung học phổ thông, kỹ thuật viên cho đến lập trình viên không chuyên. Mong rằng tài liệu tự học này sẽ hổ trợ bạn đọc tự trang bị cho mình một nghề mới trong tương lai.

Nhà sách **VĂN LANG**
25 Ng.T.Minh Khai Q.1-ĐT:8242157-Fax:8235079
09 Phan Đăng Lưu Q.BT-ĐT:8413306
E-mail: minhtri.com@hcm.vnn.vn



8935073 00868
Giá: 65.000đ